Lecture Notes in Computer Science 2468
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

John Plaice Peter G. Kropf
Peter Schulthess Jacob Slonim (Eds.)

Distributed Communities on the Web

4th International Workshop, DCW 2002
Sydney, Australia, April 3-5, 2002
Revised Papers

Springer

Volume Editors

John Plaice
University of New South Wales, School of Computer Science and Engineering
Sydney, NSW 2052, Australia
E-mail: plaice@cse.unsw.edu.au

Peter G. Kropf
University of Montreal, Department of Computer Science and Operations Research
C.P. 6128, succ. Centre-Ville, Montreal,Quebec, H3C 3J7 Canada
E-mail: kropf@iro.umontreal.ca

Peter Schulthess
University of Ulm, Distributed Systems Laboratory
James Frank Ring 0-27, 89069 Ulm, Germany
E-mail: schulthess@informatik.uni-ulm.de

Jacob Slonim
Dalhousie University, Faculty of Computer Science
6050 University Avenue, Halifax, Nova Scotia, B3H 1W5 Canada
E-mail: slonim@cs.dal.ca

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): C.2, D.1.3, D.2.12, D.4, I.2.11, K.4.3, K.4.4, J.1

ISSN 0302-9743
ISBN 3-540-00301-0 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna e.K.
Printed on acid-free paper SPIN: 10871283 06/3142 5 4 3 2 1 0

Preface

Communities are groupings of distributed objects that communicate, directly or indirectly, through the medium of a shared context. This conference brought together researchers interested in the technical issues of supporting communities.

DCW 2002 was held in Sydney, Australia, and was hosted by the Software Engineering Research Group and the School of Computer Science and Engineering at the University of New South Wales. It was the fourth in the DCW series, the previous conferences having taken place in Québec (2000, LNCS 1830) and in Rostock (1998 and 1999, University of Rostock Press).

The program committee selected 25 papers from 59 submissions from around the world. Each submission was fully refereed by at least three reviewers, either members of the program committee or other external referees. Reviewers were selected to ensure they had the required expertise for the papers' topics. Special care was taken to avoid conflicts of interest by ensuring that if a program committee member was co-author of one of the submissions, then the paper would be reviewed by program committee members or additional reviewers who were not close collaborators of the submitting authors.

In addition to these papers, the proceedings begin with a fully refereed summary chapter, entitled Open Problems in Distributed Communities, which gives an overview of the field.

This year, we were pleased to welcome as the invited speaker Philippe van Nedervelde, CEO of E-SPACES (Belgium) and X3D Technologies (USA), who gave us a fascinating presentation on Virtual 3D World Communities.

We would like to thank the members of the program committee and the additional reviewers for the thorough reviews of the submitted papers, the authors for submitting their contributions, and the authors of accepted papers for their collaboration in preparing these proceedings. Special thanks goes to Blanca Mancilla, the local arrangements chair, for all the work she did before, during, and after the conference.

We are convinced that this conference series is very timely, and meets the needs of many researchers interested in building networked communities. We hope that you find the proceedings interesting and stimulating, and hope that you will join us in Ulm in Spring 2004 for the next DCW conference.

September 2002

John Plaice
Peter Kropf
Peter Schulthess
Jacob Slonim

Organization

DCW 2002 was organized with the support of the School of Computer Science and Engineering and the Software Engineering Research Group at the University of New South Wales, Sydney, Australia, in cooperation with ACM, ACM SIGPLAN, and ACM SIGWEB.

Program Committee

John Plaice, University of New South Wales, Australia (Chair)
Jacob Slonim, Dalhousie University, Canada (Co-chair)
Peter Schulthess, University of Ulm, Germany (Co-chair)
Peter Kropf, University of Montreal, Canada (Co-chair)

Nabil Abdennadher, University of Applied Sciences, Geneva, Switzerland
David Abramson, Monash University, Australia
Bill Appelbe, RMIT, Australia
Gilbert Babin, HEC, Montreal, Canada
Ian Burnett, University of Wollongong, Australia
Franck Cappello, CNRS-LRI, France
Brahim Chaib-draa, Laval University, Canada
Vijay Chandru, Indian Institute of Science, Bangalore, India
Dieter Fensel, Free University of Amsterdam, The Netherlands
Manolis Gergatsoulis, Demokritos Institute, Greece
Andrzej Goscinski, Deakin University, Australia
Chris Johnson, Australian National University, Australia
Vipul Kashyap, Telcordia, USA
Pierre Kuonen, University of Applied Sciences, Valais, Switzerland
Spyros Lalis, University of Thessaly and ICS-FORTH, Greece
Melfyn Lloyd, CRC Distributed Systems Technology, Australia
Mehmet Orgun, Macquarie University, Australia
Paul Roe, Queensland University of Technology, Australia
Panagiotis Rondogiannis, University of Athens, Greece
Monica Schraefel, University of Toronto, Canada
Gabriel Silberman, IBM TJ Watson Research Center, USA
Kuniyasu Suzaki, AIST Tsukuba, Japan
Herwig Unger, University of Rostock, Germany
William Wadge, University of Victoria, Canada
Andrew Wendelborn, University of Adelaide, Australia
Johnny Wong, University of Waterloo, Canada
Yelena Yesha, University of Maryland, USA
John Zic, Motorola Research, Australia

Table of Contents

Languages for the Web

Adaptive Distributed Systems

Open Problems in Distributed Communities

John Plaice[1], Paul Swoboda[1], Jacob Slonim[2], and Michael McAllister[2]

[1] School of Computer Science and Engineering
UNSW SYDNEY NSW 2052, Australia
{plaice, pswoboda}@cse.unsw.edu.au
[2] Faculty of Computer Science, Dalhousie University,
Halifax, NS, Canada
jacob.slonim@dal.ca, mcallist@cs.dal.ca

Abstract. The usefulness of the notion of distributed computation as a mere collection of self-sufficient individuals, sharing little more than the data on which they operate, is drawing to a close. With the introduction of myriad differing platforms, wireless devices and interactive protocols, the level of cooperation between participants must increase, in order for work to be organized and effected, on the fly. Distributed communities, literally groupings of individuals sharing a pervasive context, are a necessary mechanism for resolving the general problem of the lack of efficiency inherent in the rigid and atomic mode of cooperation which is the norm in distributed systems today.

1 Introduction

Distributed communities are evolving associations of participants, be they people, physical devices or software entities, mediated by a shared context. Having such a context enables much richer and much more varied forms of interaction than does point-to-point communication; as a result, distributed communities form a new topology of networked computing that subsumes different competing topologies, such as client-server, Web-based and peer-to-peer.

The community-centric approach to computing is increasingly desirable. In the last few years, there have been numerous conferences, workshops and journal special issues on *ubiquitous*, *pervasive*, *context-aware*, *location-aware*, *mobile* and *adaptive computing*. Communities offer an approach that answers many of the general questions raised by all of these disciplines.

A community is a medium of collaboration that completely transforms the participants involved; it is *not* simply a space for the random interaction of individuals. Through interaction with a structured community, instead of restricted interaction between individuals, the processes of acquiring and disseminating information are made simpler, more fluid, and ultimately, more efficient.

As an example, if the notion of distributed community is applied to the software driving the devices used for automatic navigation of an automobile down a highway, then the focus of a component's operation changes: it is no longer attempting to build a complete world view for itself; rather, the world view is built collectively by the different components.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 1–9, 2002.

This paper proposes that distributed communities are consequently *necessary*; they adapt and support ways in which we naturally communicate, rather than having individuals adapt to the operation of individual technologies. For inspiration, we begin with a brief discussion of communities from outside the computing world. We then elaborate on three key aspects of community computing: interactive context, persistence of interactions, and associated networking issues. For each, we attempt to raise open questions that are still not fully resolved yet, and suggest ways in which they can be approached. In addition, we give a brief overview of the papers to be found in this volume, and show how the research described therein is relevant to the problems outlined in this paper.

2 Distributed Communities in the Real World

Communities as a mechanism of efficiency are a naturally occurring phenomenon. The eukaryotes, complex bacteria with internal division of labor (mitochondria, nucleus, etc.), came to being billions of years ago through the creation of communities of prokaryotes, simple bacteria. This development allowed for a much more efficient use of energy and ultimately led to the development of multi-celled individuals, literally cell communities, which are themselves more efficient in energy use and production than their precursors. On a grander scale still, we see that the phenomenon of community has recapitulated as entire species of animals now naturally organize themselves into packs or herds as a simple matter of survival.

This trend, with communities formed of cooperating individuals being more efficient than the previous collection of randomly interacting isolated individuals, has continued to today. For example, if we look at the development of the factory in the last hundred years, it has progressed from the completely atomized world of the Taylorist assembly line to the team work of Toyota's car assembly line. In the Taylorist model, caricatured in Charlie Chaplin's *Modern Times*, each worker on the assembly line has exactly one task, and when the line breaks at some point, a new group of workers comes in to fix the line. In the Japanese model, workers are grouped in teams and work together on much larger entities (for example a motor rather than simply one or two bolts), and all are capable of doing each other's work; furthermore, if the assembly line breaks at their point of production, they are responsible for fixing it. Factories using the Japanese model of production are typically much more productive than the Taylorist model, because they benefit not just just from the workers' hands, but also from their intellectual capacity, as manifested through their interactions and discussions.

3 Communities in Software Engineering

In the world of software engineering, the trend of increasing productivity through the use of communities has been taken one step further with the open-source concept. Thirty to fifty years ago, we focused on proprietary software for standalone computers and on special-purpose programs such as operating systems

and compilers. Now there are hundreds of open-source activities, including the huge GNU, Linux, Gnome, KDE and Mozilla free software projects, which have brought together thousands of programmers, documentation writers, Web site designers, and others, to produce industrial-quality software whose bugs can be corrected far more rapidly than using any other approach: both the projects and the resulting programs are more flexible.

This concept can be extended further, from the cooperation of individuals developing software, to the operation of distributed software itself. If we as individuals can reap benefit from interaction and cooperation, it seems only natural that our software should be able to do so as well. Already, huge benefits have been realized by employing effective communities such as networks of computers, shared memories and multi-process distributed systems. Additionally, in moving from distributed objects to distributed (operational) components, more and more meta-information is being added to enable the seamless integration and interaction of components. We have reached a point where distributed community-centric computing becomes feasible.

But just what is it that makes the community more attractive? Quite simply it is the existence of a *context*, itself dynamic, shared by the members of the community. The context is the set of shared rules and mannerisms, implicit or explicit, common history, and so on, that mean that the members do not have to continually rediscuss the most trivial details of interaction. For computing objects, the context can initially be considered to be a dictionary and/or universal catalog, a set of attribute-value pairs shared by the members of the community. As these values change, either through outside forces or through the actions of the members themselves, all of the members can sense the change of context and adapt their behavior accordingly with the dynamics of the interaction.

4 The Role of Distributed Context

A shared context is, in a nutshell, a framework for the creation and development of communities. In other words, a community can be thought of simply as an abstract logical association, based on interaction through a shared context.

One key element in the development of community is the idea that communications must not be guarded, focused on a single protocol, and without the potential for multiple participation. Many systems such as commercially available publish/subscribe middleware systems approach this problem through distributed messaging, but often end up (or actively seek to become) the only means of interaction and the technology dictates the interactions. A shared context must reverse this scenario and allow the interactions to dictate its content or behaviour. The shared context must be used as a medium that directs and negotiates for further communications.

In a sense, a context is the mutual state, the "present moment" shared by a community, but, as with the universe at large, is clearly the result of an accumulation of such moments. The context can act as both an object of the community's manipulation, and as a means of interaction. Thus, as members communicate,

the context of their interactions grows, as does the community as a whole. Further, if the context is used as a means of facilitating other communications between participants, it is a given that structures for improved communications will evolve *within the context*, making it a powerful, if not essential tool for the coordination of general transactions within the community. In a grander sense, the community's entire mode of behavior (its *culture*), will evolve in the context. This is perfectly analogous to the physical world, since our notion of culture is generally that which is manifested through our interactions with one another.

Nowhere is a need for shared context more obvious than in wireless applications. In the wireless domain, the interaction of mobile devices with fixed or other mobile services will inevitably require shared context, with well-known and understood dimensions. Currently, this functionality is built into local transport mechanisms, mostly notably in Bluetooth's service discovery layer, but this is a myopic approach to a context protocol that must have a near-ubiquitous scope. The context should permeate not only the realm of interaction between devices, but every layer of implementation within the service infrastructure, and certainly within device software.

But wireless applications are by no means the only domain of application. In the corporate, administrative and military worlds, the use of context can be directly extended for the purposes of security: When the passive-protocol approach is inverted, giving control over participants to the context between them, bandwidth-intensive activities such as constant authentication become much more efficient.

The emergence of communities from shared context can be thought of in the same manner as the early communities of the Internet. These evolved not just from the net, per se, but through specific services and protocols *on the net*. These include the communities of Usenet, Gopher, MUDs, etc. In a shared context, the same notion of communities would arise not from the context-sharing protocol/network, but in the *contexts shared by the network*, and the applications that used them. Implicit in this idea is the notion that many communities could exist within the same global context, their individual sub-contexts possibly overlapping, or completely distinct.

While shared context is obviously essential to distributed communities, there remains the question of how best to use it. While it is possible to have participant systems constructed entirely from context-sensitive languages, this is not a necessity. Similarly, there is no requirement that participants use the context as a primary means of interaction; the context simply provides for a useful medium of cooperation or for developing further cooperation. Otherwise, we fall back into the trap of protocol-focused behavior.

5 Common Infrastructure

In the presence of shared information, there is the question of managing this information. This is no less true of communities or contexts. The management goals of communities and contexts are different, though related. To be effective,

all management must contribute to keeping the community centred on its self-stated purpose and should be transparent.

Community management focuses on the individuals or devices that form the community and how they can interact with each other or with the shared context. One aspect is thus the management of the community membership, which includes awareness of the community and individuals within the community, ways to join and leave the community, ways (or expectations) to contribute to the community, and respect for the privacy of individuals and their data in the community. Membership management must also introduce new community members to the existing context.

Another important aspect of community management is the interactions that are possible that must be integrated into the context, whether facilitated by the context or developed outside the context. These interactions can be between communities (or sub-communities), between the community as a whole, between individuals in the community, or with individuals outside any community. Different interactions can have different purposes that are best served by alternate communication styles; they may be collaborative, informational, or social. At the level of an individual in the community, the mechanisms used for the interactions should be transparent. The individual should not be concerned with the devices or capabilities of devices, or with the location or structure of the information in the interaction.

Few, if any, communities remain static, and community management must allow for evolution. The evolution may be based in the shared context of the community or may occur independently of the context. The infrastructure that supports the community should allow for these changes and provide ways for the community to integrate the evolution back into the shared context or to adapt the infrastructure easily. As with peer-to-peer systems, it may also be desirable for individuals, sub-communities, or related communities to isolate themselves from some aspects of the community's evolution; this would need to be another aspect provided by the infrastructure that supports the community.

In contrast to community management, context management focuses more on the information that the community uses. This information can be from the community as a group, from individuals who are willing to provide advice or services based on their private information, or from legacy systems. Depending on the community's goals, protecting the privacy and access to the data may be of vital importance to context management. Moreover, access to the data and any such restrictions should remain as transparent as possible to the individuals within the community.

Context management must deal with membership issues. In the case of context, membership relates to deciphering what data or type of data is relevant to the context and can be added to the context, and how the new data is presented to or from the context. It could even be desirable for the infrastructure behind the context to present the context in different ways according to the individual, device, or relate context within the community, such as multimedia sources or multimodal inputs. For this reason we consider the context to be *pervasive*.

No successful digital community can exist without history. History, for example, is what makes Usenet so useful, allowing the revisitation of content posted since its inception. History is also one reason why the Web is best not classified as a community, but as an infrastructural window to communities with persistence, ranging from chat rooms to Web logs to Usenet portals.

With shared context, the current state is always accessible as the aggregation of all dimensions with values. It is completely unreasonable to demand of a vast, distributed context network that absolute persistence of all context changes be maintained at all times. The performance limitations imposed by the overhead of collecting, serializing and archiving this data would destroy the utility of the network. Instead, in keeping with the independence proffered by peer-to-peer systems, the onus of context persistence could be placed on the specific community, and the community itself need not archive all of its own context changes. Indeed, changes to the value of certain dimensions could be monitored as indicators that a snapshot of some subset of dimensions be recorded. Certainly, different communities would need to archive history in different ways.

With shared context comes the necessity to be able to secure and protect the context. The notion of "security" often encompasses three separate aspects. First, basic security covers the mechanisms used to keep information private such as encryption algorithms or protocols. Second, privacy encompasses the policy end, knowing what data can be shared based on the user's desires. Third, trust entails having the users of the system believe that their stated wishes are handled appropriately or that the context information that they are presented with is correct. These three issues are separate, and each has a role to play within a community.

The technical aspects of context management will delve into mechanisms, protocols, and formats for exchanging information between individuals to facilitate their interactions. Of critical importance for the shared information is an understanding of if, how, or where this information is replicated. On one hand, replication improves access and persistence. On the other hand, replication requires consistency control. These issues are part of the concerns of distributed database and multidatabase research, but each one presupposes the existence of a transaction manager. With a dynamic environment such as the community, centralized or well-known transaction managers may not always be accessible.

As communities evolve, so must their contexts. When a community evolves outside of its context, the community is responsible for drawing the changes back into the context, and systems that support communities should account for this. When a community evolves within or through its context, changes to the context may originate with the individuals, with a particular interaction that characterises the evolution, or even with self-adaptations that are derived from sequences of interactions.

6 The Significance of DCW2002 Contributions

During the *DCW2002* conference, several of the above research themes were addressed, either explicitly or implicitly. In this section, we examine some of the papers and relate them to the above discussion.

6.1 Position Papers

The position papers session was, as planned, very lively, with intense discussion. It was intended that the authors present strongly held views, even if the technical issues had not been fully resolved.

Bill Wadge (p. 10), with his paper ' "The Medium" is the Message', presents the views that most closely correspond to this work's theme: what defines a community is the pervasive shared context or medium; in his paper, he shows that to properly address this theme will not be easy, since both mathematics and software engineering have, through the years, done their utmost to empty the medium.

Gavin Brebner *et al.*(p. 15) address the problem of how to access user profiles, while securing privacy, to configure devices that are not continuously connected to a single, centralized server. This is a special case of a much more general problem with real communities, which are, in practice, never fully nor permanently connected.

Holger Christein *et al.*(p. 24) and Rima Saliba *et al.*(p. 35) develop adaptive models for two key issues for all communities, respectively, *presence awareness* and *security*. We believe that the context should inform both of these, and that the behavior of each of them should depend on the context.

6.2 Adaptive Networks

Adaptive networks are a special case of community-based computing, a self-configuring medium of interaction. In this section, several papers show that having adaptive networking substantially increases the overall throughput.

Xueyi Wang *et al.*(p. 42) present a Site-based Reliable Communication Protocol, where the member sites of a collaborative system dynamically choose how to communicate with their neighbours. Yoon-Jung Rhee *et al.* present two algorithms for differentiated Web service; one is based on high- or low-grade service (p. 54), the other on Web server logs (p. 64); in both cases the throughput is increased. Shui Yu *et al.*(p. 74) focus on application-layer anycasting for IPv6, and show that a requirement-based probing algorithm improves results. These are just a few examples that show that if we can consider the activities of a community's participants, then the entire community's actions can be optimized.

Jean Vaucher *et al.*(p. 84) take a different approach. They examine the possibility of using a free peer-to-peer distributed system as the basis for an informal distributed community. Their experimental results are negative, which continues to suggest that only by using an explicit context can communities be formed and held together.

6.3 Collaborative Systems

Collaboration is at the heart of the community-based approach. The papers in this section, although they do not all use an explicit context, give a number of examples of comprehensive views of a distributed system developed and shared by the participants of the system itself.

Tim Moors (p. 100) presents a mechanism to enhance interactive group audio with a control channel, used for speaker identification, feedback and turn taking. Alberto Morán *et al.*(p. 113) show how the use of collaborative spaces, with specific services tied to these, facilitates collaborative writing. Both of these examples use a context to bind the community of speakers or writers together, as well as to enhance their interaction.

The other papers focus on the technical means for ensuring compatibility of the differing views of the participants in a system. Liyin Xue and Mehmet Orgun present two papers (p. 125 and 138) on the intricacies of managing inconsistency in the actions of the participants, and how to reconcile the differences.

Two papers give rule-driven approaches to defining software agents. Murthy and Abeydeera (p. 151) present a specific algorithm for e-marketing negotiation; Morad Benyoucef *et al.*(p. 165) give an experimental study of different negotiation strategies, simply by modifying the interactions between the agents.

Finally, Theodore Chiasson *et al.*(p. 182) present an approach to managing distributed information in a process-oriented peer-to-peer environment without relying on a centralized transaction manager.

6.4 Languages for the Web

Language — and communication in general — is the most fundamental component of community, since how we can express things transforms what is expressed. At the same time, the act of communication alters the participant, and the community's context transforms communication. One would therefore expect language development to focus both on content and context.

Two papers focus on improving Web-based 3D support. Guillaume Potard and Ian Burnett (p. 193) define a markup language for encoding 3D interactive audio scenes. Boon-Hee Kim *et al.*(p. 204), take a different tack, extending VRML to a real programming language, with database support, for programming Web-based 3D-objects.

Dieter Fensel (p. 215) relates the history of developing a logical metadata framework for defining the Semantic Web; this turns out to be have been an arduous task, since it is difficult to express anything without referring to the context. Ioannis Kassios and Monica Schraefel (p. 228) take the opposite approach, and focus explicitly on the context, presenting a new language for defining context-aware hypertexts.

6.5 Adaptive Distributed Systems

Any successful community must be able to adapt to changes in working conditions. This section's papers give examples of distributed systems that adapt to a number of different kinds of change.

Tim Mansfield *et al.*(p. 240) present a Social Portal, a series of Web pages — along with the related infrastructure — that are presented in a different order depending on the recommendations of the people viewing them. Till Schümmer (p. 253) looks at a similar problem, this time for building communities at Web shops, while Seng Wai Loke (p. 266) focuses on the services that might be enabled as a user is moving from one location-based e-community to another.

On another level, Anthony Howe and Mantis Cheng (p. 278) show that several redundant video sources can be used to ensure quality of service to many users, even under serious network perturbations. Finally, Peter Rigole *et al.*(p. 291) look at integrating a number of devices in an automated home, so that the result is coherent.

7 Conclusions

Distributed communities are the next natural step in the development of distributed computing. Supported by an active and pervasive context, the participants are able to interact in much more efficient ways than they could using a simple atomistic view of computation.

However, for distributed communities to become widely accepted, much work will have to be undertaken, at all levels. At the conceptual and implementation levels, it may well be that a few ideas and primitives suffice to enable an impressive infrastructure; consider, for example, how just HTTP and HTML were sufficient for the development of the Web. However, at the social level, in the sense of our interactions with computing devices and with each other, the implications are still difficult to grasp.

"The Medium" Is the Message

Bill Wadge

Department of Computer Science
University of Victoria
PO Box 3055, STN CSC
Victoria, BC
Canada V8W 3P6
wwadge@csr.uvic.ca

Abstract. It is our position (following Plaice and Kropf) that mathematics and software engineering does not provide a very good basis for understanding communities. These formalisms are necessarily influenced by a mechanical, atomistic outlook which sees collections as arbitrary assemblages of self sufficient individuals communicating point to point in a vacuum.
I suggest instead that ideas of Marshall McLuhan provide a much better starting point because they give a central role to the medium by which the members of a community communicate. Furthermore, I argue that the phrase "the medium" should also be interpreted in the sense used in (say) biology, as a nurturing extended substance (a plenum) which fills the space between the individuals and to some degree permeates their interiors.
Finally, I argue Swoboda's context server project can be understand as realizing Plaice/Kropf "intensional communities" by formalizing and implementing a nurturing medium which permeates the (software used by) each community member.

1 What Is a Community?

It goes without saying that DCW work requires an understanding of what exactly is a community. However, this question is not one that can be settled formally, because communities inherently involve people. A Web community is a kind of embedded system, but with people rather than processors on the inside. To understand communities on the Web means some understanding of pre-Web communities, which can be traced back to the very origins of mankind.

If DCW ever decides to adopt a patron saint, I nominate Marshall McLuhan, the late Canadian media theorist [1]. It was McLuhan, of course, who coined the term (electronic) "global village". It's easy to forget that when he introduced the concept, in the sixties, the Internet did not exist, the first few communication satellites had only just been launched, and most television was still black and white. It was only thirty years late, with the arrival of the Web, that we can understand just how well he grasped the true community-building nature of electronic media.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 10–14, 2002.

But more generally, he was among the first to understand the real importance of media in general, to state clearly that media are not just passive conveyers of messages. He argued convincingly that the media can play a vital role in determining the nature of society. McLuhan had communication media in mind, but the principle is equally valid for more solid forms of exchange, such as physical transportation.

The development of the automobile industry throughout the twentieth century provides a striking confirmation of McLuhan's theories. They were originally introduced as an alternative to the horse and buggy, and the first customers were typically well-off doctors. Then farmers used trucks to bring produce to market. A few decades later the auto industry was central to the economy and the network of paved roads permeated the whole continent. During WWII internal combustion engines made possible the large scale destruction of cities, and then caused them to be rebuilt along new designs (suburbia, freeways, shopping malls etc). The very nature of modern society is embodied just as much in the car/road system as in the presumably more important homes, schools, and factories which this system connects.

2 Communities in Mathematics and Engineering

It's useful to contrast the McLuhanite view of community with analogous concepts in mathematics and engineering.

The corresponding mathematical concept is the set. Sets seem simple to us now, but the current notion took many centuries to evolve. A modern set is an arbitrary collection of mathematical objects, and it is the very arbitrariness which proved difficult to master. The elements of a set may have nothing in common, other than the fact of being ... elements of the set in question. Furthermore, there is nothing which holds them together, other than the curly parentheses which fence them in. This may sound more like social science than mathematics, but in fact many serious (and precisely formalizable) foundational issues can be understood as symptoms of the 'unnaturality' of the arbitrary set. The Axiom of Choice is the main formal assumption which assures the existence of such an arbitrary set, and it proved to have many surprising and unpleasant consequences. Tarski and Banach, for example, used it to prove that a sphere can be divided into a finite number of parts that can then be reassembled into two spheres - of the same size [6].

Of course, I accept that mathematics is indispensable in the effort to build and understand Web communities. But mathematics alone is not a reliable source of insight into the real nature of communities. It only works because, over centuries, it's been purged of these aspects.

Computer scientists have also produced notions analogous to community, in several application — in particular, the OO paradigm and in abstract approaches to concurrency. These artificial communities are more realistic than the abstract sets of mathematics, because they are dynamic — they carry out computation and communication. But they still share with mathematics the basic atomistic

assumption that a community is the sum of isolated individuals. Plaice and Kropf [4] point out that the OO paradigm involves atomic, self sufficient individuals sending atomic messages in a vacuum — that "there is no context; outside the objects there is nothing".

The current concurrency models make exactly the same atomistic assumptions. Robin Milner, in his Turing Award lecture [2], admits shared memories (a rudimentary form of context). But they are treated as just another process, so that the model is strictly peer-to-peer. He thereby excludes an all-pervading medium (a plenum) with properties essentially different from those of an atom.

3 Humans as a Paradigm

Where can we find paradigms for human communities on the Web? The best source is off- or pre-Web human communities, and McLuhan's work is an excellent starting point. But there is another paradigm, literally under our noses: the human body itself.

I invite any reader who knows even a little of both anatomy and software engineering to reflect on the design of the human body. The body's design, by traditional engineering standards, is very poor indeed. To some extent, it is modular - the body is composed of organs - but in many ways it violates information hiding and separation of concerns. An individual organ may have several distinct purposes (like the bones, which provide the basic framework but also generate the blood supply). Some tasks are shared by a number of organs, and others (such has heat regulation) are distributed throughout the body and are not the responsibility of any particular organ.

More significantly, every organ in the body is permeated by at least one of the body's many systems. The blood supply enters every single organ and even (via cell membranes) inside individual cells. The nervous system also spreads through most of the organs. In a sense, if we regard the body's organs as modules, their interfaces are anything but narrow.

The body is literally much more than the sum of its parts, because these systems are not really separate parts. They are media which weld the parts together, nourish them and allow them to communicate. The body is a community of organs and the nerves and blood vessels are its highways and Internet.

Moreover, the human body itself has evolved to function as part of larger communities. Human languages allow much more effective communication and sharing than anything found in the animal world. But all man's other senses are well adapted to communal activity. In the words of McLuhan's teacher and collaborator, Walter Ong [3, p.325]:

> The openness of living things is most spectacularly evident in man. Through his respiratory, transpiratory, and digestive organs as well as through his senses, man interacts, as do lower forms of animal life, with a great deal of the world around him. But man's interaction goes infinitely further. Through his intellect, man is open to absolutely everything, though he may have trouble making contact with it all, and indeed will

> always be far from such an ultimate achievement. Man is the most open system we know of.

Humans are most of all open, in the final analysis, to other humans; but, for the most part, not directly. They are open to the various media (the air, food, sound, speech, vision, ideas) which hold human communities together.

4 Contexts as Media

My position is that one of the main challenges facing the engineers who build infrastructure for communities is to design machines (computers, software, networks etc.) which further the members-in-a-medium model. The challenge lies in the fact that machines are, by default, mechanical: they are the sum of their isolated parts. The task is not impossible; for example, the Internet has evolved (and not mainly by design) to resemble more and more the blood and nervous system of the human body. The tendrils of the Internet are rapidly penetrating every area of human activity. With the emergence of wireless networks, the Internet will literally physically permeate human society and even the human body.

For those producing software to support distributed communities, the goal is to make the applications as far as possible open/permeable like the people who use them. This, I suggest, is the best way to understand Plaice and Kropf's "intensional communities" and the related tools.

Programs in an intensional language (such as ISE) execute in a context (possible world) which originates outside the program yet, by default, permeates all the components of the program and is available on demand at any point. Inside ISE, components can alter the context and so use it as a kind of bloodstream to coordinate widely separated activities.

The aether server project of Swoboda, Plaice *et al* [5] (still under development) can be understood as supporting the same kind of intensional medium outside any particular program or machine. (In my experience, the technical details of [5] are clear enough but people have trouble understanding the real purpose. I hope this paper helps.) The aether server is essential infrastructure for supporting the intensional model. In fact, if McLuhan is correct, this medium is the real message of the intensional communities project.

References

1. Marshall McLuhan. *Understanding Media.* MIT Press, Cambridge, Massachusetts, 1964.
2. Robin Milner. Elements of Interaction (Turing Award Lecture). *CACM* 36(1):78–89, January 1993.
3. Walter J. Ong. *Interfaces of the word: Studies in the evolution of consciousness and culture.* Cornell University Press, Ithaca, 1977.
4. John Plaice and Peter Kropf. Intensional Communities. In *Intensional Programming II*, pp. 292–296. World Scientific, Singapore, 2000.

5. John Plaice, Paul Swoboda, and Ammar Alammar. Building intensional communities using shared contexts. In *Distributed Communities on the Web*, *LNCS* 1830:55–64, Springer-Verlag, Berlin, 2000.
6. Stan Wagon. *The Banach-Tarski Paradox*. Cambridge University Press, 1986.

A Middleware Architecture for Personalized Communities of Devices

Gavin Brebner, Yves Durand, and Stéphane Perret

HP Labs Grenoble,
5 avenue Raymond Chanas – Eybens, Grenoble, France
{gavin_brebner, yves_durand, stephane_perret}@hp.com

Abstract. This position paper outlines our vision for a world of user devices not continuously connected to the Internet. Our first concern is to ease the usage of these devices by adapting their services to the user with data from a local profile. Maintaining the privacy of these user data is a major requirement that impacts all the work we do around collecting and exploiting these data. A major problem is to ensure the visibility of these data in a volatile environment.

1 Introduction

As Internet users, we provide personal information to a growing number of service providers with little or no control over its usage, and no means to properly track subsequent access of this information. Some companies have recently made announcements proposing to handle our personal information centrally, offering the possibility of a unified repository [1], but raising additional trust and privacy concerns [2]. We have chosen to investigate an alternative to this trend by storing personal information on client devices, increasing the possibility of putting the user in control of his or her personal information.

HP Laboratories Grenoble is working on communities of end-user devices that are not continuously connected to the Internet. A community of end-user devices denotes appliances that are able to aggregate spontaneously to offer personalized services to the user. In this paper, we identify some of the problems that arise from using multiple devices to help end-users. We propose a layered software approach to remove complexity from the user and we identify the data availability and privacy issues caused by the volatility and mobility of such a community of end-user devices.

2 Position – Constructing the Vision

We work towards a vision of a human being who is helped by client devices that act as interfaces into the digital world. These devices collaborate with each other, and use dynamic aggregation to maximize the functionalities available to the user, while keeping the user experience as natural as possible. We meet two difficulties in trying to insure this desired simplicity: First, the environment of

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 15–23, 2002.

these devices is fundamentally dynamic: devices arrive in a community, interact with others, and leave. Second, personalized services require an important amount of information about the user and its environment. There is therefore a certain amount of work to do to create the desired illusion of simplicity for the user.

The notion of context is fundamental to our vision: it permits actions to take place with only minimal user interaction, and without the user being required to be knowledgeable of all aspects of a system. An important aspect of simplicity is reducing the amount of information a user needs to know. The context data is basically composed of all the information gathered to compensate the illusion of simplicity for the user in order to enable personalized services.

Personalization is still very much seen as a vertical, application driven, activity. Efforts towards more horizontal distribution of information, a.k.a. "intermediatory services" have led to centralized, web-based information services. In contrast, our main concern is to make the contextual data available to chosen devices that are mobile and volatile. This guarantee on availability is not the only constraint: we must also guarantee privacy and some kind of consistency on these data.

Section 3 presents the model that drives our vision. Section 4 details the architecture we are proposing to implement the problem. Section 5 presents the issues we are facing regarding contextual data management. Finally, Section 6 presents relevant experiments we did that give us confidence in our vision.

3 Model – The Personal Data Store

We present contextual data as a single complex data structure managed automatically by the user devices regarding change in the environment. The figure 1 illustrates the model centered on user.

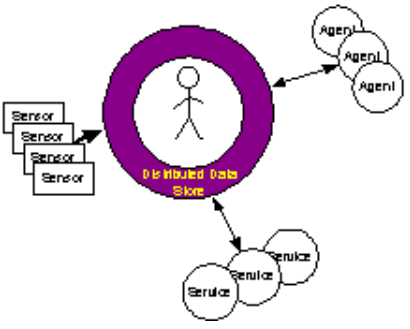


Fig. 1. Personal Data Store

Contextual data is kept in a personal distributed data store running on user devices. Three components interacts with this data store:

Sensors: produce raw data on user and environment
Agents: compute inferred data on behalf the user
Services: access user data to help improve user experiences

Inferred data are calculated from raw data: agents installed on one or more devices are capable of analyzing the raw data, and transforming it into information and knowledge. Raw GPS coordinates, for example, can be stored, and then used by a specialized agent to generate information about the likely environment (at work, at home) providing the agent is available. We envisage the generation of such information as being a permanent, dynamic process, with any data changes being reflected as quickly as possible by updates to the generated information. We see the role of information generation as being done by agents rather than services, as it will often require detailed local knowledge of the environment, and be a highly personal, and therefore private operation. These agents may, of course, be supplied by a service, and / or make use of local services as appropriate.

Our vision differs from Microsoft's .NET, where all such data is stored on a central remote web service. However, we expect to work in parallel with such systems by linking data in remote data stores with that in the local personal data store, and by permitting services direct access to the personal data store where possible. We see access to personal data without passing through a centralized web service as a critical privacy protection measure. It is also vital if data is to be available even when a permanent high-performance connection to the Internet is not available; as we envisage locally available services and applications, this is important.

The Personal Data Store has several responsibilities towards context data:

Unified view: It presents a common input / output interface for data. Thus, "sensors" that can interact with the appropriate API can be easily integrated into a (potentially) complex system.

Compartmentalization: The personal data store acts as an enabler for the distribution of data, if this is appropriate for the particular application. Parts of the system outside of the data store do not need to be aware of distribution.

Storage management: The personal data store is an ideal place to cache data. Caching may be required for performance, or simply to cope with intermittent connectivity.

Access control: Personal data must be protected from abuse. The use of a common store is an enabler for the application of privacy protecting technologies.

Trust management: As devices are there to help the user, it is important to build a relation of trust with him or her, and for the device to have all the data necessary to be useful. In turn, Devices will have relationships with

other devices. We qualify these relationships with trust levels. Friends can be considered trusted entities that will collaborate fairly with the device, relations will be descended from the same source, and may share certain basic characteristics e.g. adherence to certain protocols.

The personal data kept in the data store can be used in a number of ways to enhance the user experience. We envisage two solutions here: standalone agents located on client side on one hand, and adapted services, located remotely, on the other hand.

3.1 Personal Assistant Agents

Our agents guide a user by providing hints and assistance. Access to relevant knowledge of the local environment is key; such agents need to know the meaning behind terms such as "here", "at home", "my PC". We envisage such agents being located on any device appropriate for the task in hand, or being made available on the fly by other devices or services. Examples of the use of personal assistants are proxies for e-commerce, information searches, device trouble-shooting, and messaging.

3.2 Services That Use Personal Data to Be More Effective

More conventional web services will exist. These services may be available across in the Internet, or may be available more locally on a device accessible within the local ad-hoc network. These services will have means of accessing personal data on users to be more effective; the client device may provide information when calling the service, or the service may in turn ask for information from an information source. That source may be the calling device. Web services will be able to offer the advantages arising from the centralization of multiple requests, but at the expense of reduced user privacy, as data is exposed to a third party. Web services and personal assistants may well inter-operate, with the personal assistant acting as a front end to a service, guiding the user in the choice of service, and limiting the exposure of data.

4 Proposal – A Layered Software Approach

We have structured the problem in three layers. Each layer implements several mechanisms with explicit responsibilities. The figure 2 illustrates this architecture:

We call the lower level that sits on top of the transport layer, the "Aggregation and Connection" layer. This layer ensures communication between remote "objects", i.e. executables, services, etc. Our intention is that this layer frees the layer above from the burden of managing references: for example, it avoids blocking when accessing a data from a resource that just moved. We also implement in this layer fallback mechanisms that define alternatives when a resource fails.

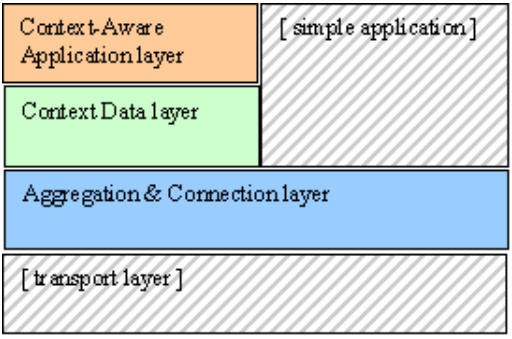


Fig. 2. Multi-layer application model

The intermediate layer called here "Context Data" layer implements the basic mechanisms for accessing consistently the context data. It provides a unified view to the data, and standard handles to access and modify it. It implements compartmentalisation, access control, and privacy protection. It provides storage facility, such as caching.

The upper layer, namely the "Context-Aware Application" layer, is responsible for using the data, for example for improving accessibility for the user. It may be implemented by applications facing the end-user, or by stand-alone service used by other "simple applications" through the aggregation and connection layer.

In the remainder of this document, we detail the functionalities and implementation challenges that derive from this architecture.

5 Context Data – Availability and Privacy

The contextual data is heterogeneous by essence; it may have multiple different repositories (i.e. local on device, local in the community or remote from a web service). The interface to the proposed middleware does not make this explicitly visible, although meta-data may be used to influence the distribution of data. Figure 3 depicts several types of data to illustrate situations.

Due to the ad-hoc nature of the device community, and the likelihood of frequent disconnections, the contextual data updates introduce tough consistency issues [3]. We face a well-known problem of trade-off between consistency and availability. In presence of network isolation, it is not possible to enforce strong consistency between operations on data. We need basically to relax some of those idealistic requirements: anytime availability with strong consistency. A loose coherence, such as proposed in the shared memory system Munin [4] is a good example of how to tackle this issue for migratory objects.

The "Context Data" layer is responsible of this issue. It relies on the properties offered by the infrastructure middleware: We have chosen to consider initially

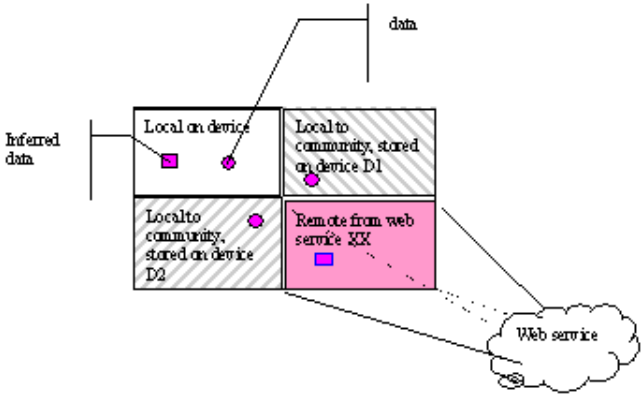


Fig. 3. Types of Personal Data

that it offers a strictly coherent model for distribution across multiple-clients, and have taken an approach to trust that restricts the permitted location of data items based on data sensitivity, device capabilities, and a user-specified trust policy. This allows us to deal with the heterogeneity of device security capabilities. The implementation of these models uses a coherence protocol based on a write invalidate protocol with dynamic and conditional caching rights.

The "Aggregation and Connection" layer provides an abstraction for remote data access and invocation of remote objects. In other words, it considers all remote objects as abstract references that are always connected and identifiable. In order to implement it, we rely on classical "object brokering" mechanisms to abstract references to data and executables. The necessary infrastructure to implement these mechanisms is fully decentralized to cope with the volatility of our network of personal devices: the community implements a kind of collaborative and dynamic assignment of infrastructure functions. This connectivity middleware provides also a limited support for resilience: in case of a failure, it will be able to detect the alea using failure detectors and use a fallback as an alternative for the missing function, according to a fallback policy. The infrastructure present on all devices will transparently manage these "fallbacks" according to events, and relieve the layer above of this task.

6 Previous Experiments

We have recently prototyped pieces of the vision that helped us have confidence with the whole picture.

A first project called VAL (Virtual Assistant on-Line) [6] aims at simplifying the PC user experience to get in touch with PC related web services such as an accessory service or driver service. Basically, the application runs on a PC as a local web service. It makes use of two sensors: a PC sniffer that provides

information about the local system and a service manager that maintain the list of available services. Users can submit requests in natural language to this Assistant application. The Assistant interprets the request and manages to put the user in touch with the relevant service using data exposed by theses sensors. This is a good example of client side personalization that simplifies the user experience.

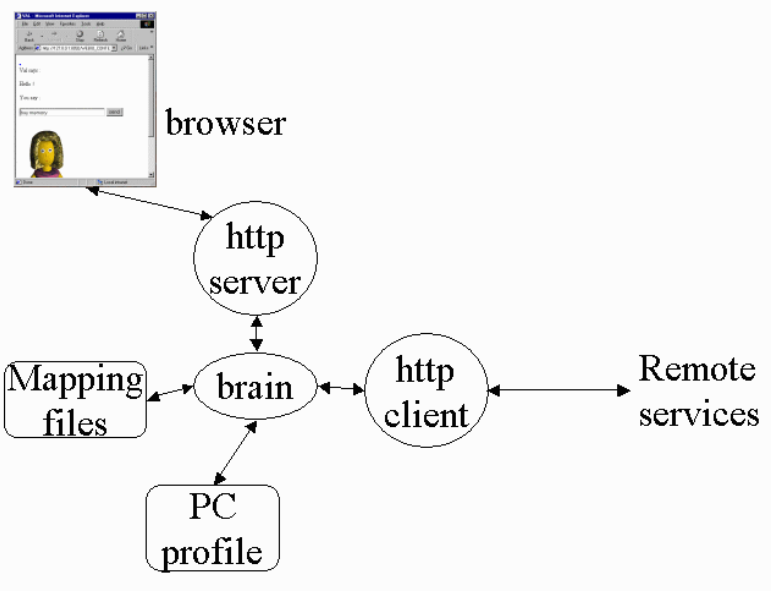


Fig. 4. VAL architecture

A second project called ICE (Instant Conferencing Enabler) aims at simplifying document sharing for the mobile worker visiting conference rooms and colleague desks. Mobile users devices such as PDA make use of two sensors from the HPL Cooltown project [5]: a beacon listener that gathers location signals (URL of a place web page), and a presence manager that exposes entities profile (URL of a user web page). A Service Broker application on the corporate Intranet has some information about the corporate environment of multimedia devices. When visiting a place, the mobile user device detects the location signal. A client application running on the mobile user device, namely the Place Walker, is able to submit queries to the infrastructure Service Broker. These requests automatically include contextual data (the user profile URL and the location URL).

The Service Broker makes use of this information (listed in the web pages) to respond to the queries with the relevant personalized service agent. This is a good

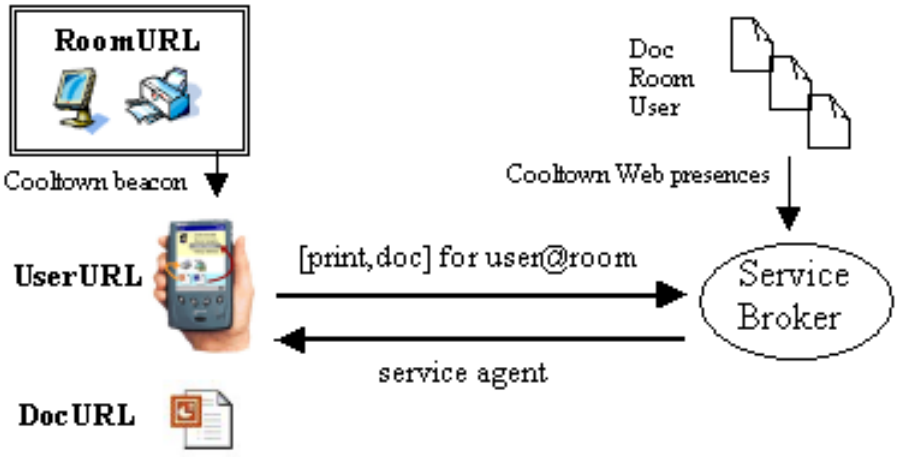


Fig. 5. ICE principles

example of context-based service brokering that simplifies the user experience as well.

Regarding the model, VAL and ICE personalized services do not use a common data store to get contextual information, but rather directly make use of well known sensors. The impact of these service was huge on users that loved simplicity. These experiments showed us confidence about the vision and the architecture we are proposing centered on a unified context data layer to enable simplicity both for user and service providers.

7 Conclusion

We have outlined our vision for user-centered device communities, indicating the direction of our current research towards this vision. We have prototyped applications that make use of context data to improve the user experience, while guaranteeing privacy through a sandbox model. On the data layer, we have prototyped a component that enables transparent access to profile data, as well as caching. We are planning to integrate these components into a distributed environment based on a lightweight and implementation of CORBA to be able to cope with the problems of mobility and disconnection.

We still are improving our consistency model: strict consistency is not envisaged, and we still work on finding a good relaxed consistency model. The other difficult issue for us to solve is the integration of a trust mechanism in our data manipulation: we look for a trust model which is both powerful enough to handle everyday problems for the user and still simple enough to be compatible with our disconnection and consistency issues.

References

1. "Building User-Centric Experiences: An introduction to Microsoft .Net MyServices", September 2001.
   http://www.microsoft.com/myservices/services/userexperiences.asp
2. Privacy@net. http://privacy.net/
3. G. Coulouris, J. Dollimore, T. Kindberg, "Distributed Systems Concept and Designs", second edition, Addison Wesley
4. J.k. Benett, J. B. Carter, W. Zwaenepoel "Munin: Distributed shared memory based on type-specific memory coherence" in Proc. Of the 1990 Conference on Principles and Practice of parallel programming.
5. Cooltown dev. Network. http://www.cooltown.com/dev/
6. G. Brebner "Matching user needs to product offerings in a friendly way" in Workshop on friendly exchanging through the net, March 22-24, 2000.
   http://www.hpl.hp.com/speeches/techtalks/2000/brebner.mar00.doc

A General Purpose Model for Presence Awareness

Holger Christein and Peter Schulthess

Distributed Systems Department, University of Ulm, Germany,
{christein, schulthess}@vs.informatik.uni-ulm.de

Abstract. A wide range of distributed application scenarios require presence awareness. Examples include computer-supported collaborative work (CSCW), collaborative browsing, chat-/audio-/video-conferencing Systems, e-commerce, tele-teaching, to name only a few. Presence awareness provides information like the location, idendity, activities and the neighbours of someone or something who or which is present somewhere. In this paper we introduce the entities involved in a general purpose model for presence awareness. The goal is to provide the necessary services to the various applications that make use of presence information.

1 Introduction

Currently there are already a lot of applications which need presence awareness information and in the near future there will be more. Well known today's application areas are messengers like from AOL, Yahoo or Microsoft, CSCW applications or virtual 3D communities like Activeworlds. Future use cases could be location based services in the area of mobile phones like friend finder or applications which coordinate a fleet of vehicles [1]. Other examples are collaborative browsing environments [2] or e-shops enriched with presence awareness information [3] to name only a few. What all these application scenarios have in common is that they are in need of presence awareness. But up to today there is no general purpose service available which delivers such kind of information. Each application implements its own proprietary solution which extends the time necessary to develop such kind of software and raises the costs. In this paper we introduce a general purpose model for presence awareness which might be the basis for development of a belonging general purpose presence awareness service.

2 Entities Involved in the Model

We will give a short definition of the entities involved in our model. The entities *presentity* and *watcher* have initially been defined by the IETF [4]. Our definitions may differ in details from those in [4].

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 24–34, 2002.

2.1 Location

A location is a virtual or a real place where someone or something is located at or is *present*. Examples of locations in the virtual world are documents (HTML pages, text documents, ...) or hosts. In the case of a document, being present might mean that someone has opened a document. Locations can be specified here by using e.g. URLs. In case of hosts, being present could mean someone is logged on to the host. Locations in this case can be specified by using IP addresses, maybe augmented by a port number. A higher-level abstraction with location flavor might be a chat room. Examples of locations in the real world are postal addresses or GPS coordinates.

2.2 Presentity

A presentity is present at a location. Each presentity has a unique identifier. In the real world a unique identifier could be the postal address of someone. In the virtual world a unique identifier could be of the form `user@domain`. A presentity needs not to be of flesh and blood but may also be a running program, maybe a bot or a user agent. Like locations a presentity provides presence information. It is represented as a tuple in the following way:

$$P_I = \{P_D, L, S\}$$
$$P_D = \{P, D\}$$

Whereby P is the presentity to which the presence information is associated. D is a *discriminator* between multiple instances of the same presentity. A Example of use could be a user who logs on simultaneously to a CSCW Application or has several browser windows opened. In this cases the term *session* or *user session*

is often used. So P_D can be interpreted as a user in its session. L is the location where the presentity is present. S is the current *state* of the presentity. Examples of state information of a presentity are "online", "busy", "idle", "reading", "locking line 10 of document". Being present somewhere does not necessarily imply being available. Someone may participating in a meeting and do not want to be disturbed maybe by a phone call or by a secretary. Thus this person is present in a meeting but not available for people not participating. The state of a presentity might provide information if a presentity is available or not. Many application scenarios need not only presence information but also the information whether a presentity is available [5]. State information of a presentity may change very frequently, so it is transient. State information can be a tuple itself, e.g. $S = \{$"do not disturb", "sleeping"$\}$.

We define the following relationships between presentity, discriminator, activity and location (to simplify matters we omit the discriminator in the textual explanations):

- States are always bound to a presentities present at locations. It means that only a presentity present at a location can have a state.
- multipresence and multilocality: A certain presentity can be present at more than one location at the same time and at a certain location there can be more that one Presentity present at the same time:

$$(P_D)_t \leftarrow n : m \rightarrow L_t$$

- multistate: A presentity at a location can take more than one state at the same time and a certain state can be taken by more than one presentity, present at a location at the same time:

$$(P_D, L)_t \leftarrow n : m \rightarrow S_t$$

  This means that a certain state can be taken by different presentities present at the same location at the same time as well as a certain state can be taken by the same presentity present at different locations at the same time.

In addition to presence information associated with each presentity it may have *properties* assigned. In contrast to state information which is transient, properties are persistent even across sessions. Examples of properties could be the e-mail address of a user, the endpoint of a users video conferencing system or a video codec which a workstation is capable to receive.

2.3 Watcher

A watcher is interested in presence information. There are different ways a watcher can get this information: First a watcher can fetch presence information once. In this case it gets a snapshot of current presence information. Second a watcher can poll presence information at regular intervals. Third a watcher can

subscribe to presence information . In contrast to the prementioned possibilities it only gets notification (and each time) presence information changes.

A watcher can take the role of a presentity at the same time and vice versa. In many scenarios this is the normal case. Like for instance in a collaborative browsing application where each participant is present at some web page (or web site) and at the same time interested in who else is present there.

2.4 Vicinity

A vicinity is a set of locations which are adjacent in terms of awareness. It means that each two presentities which occupy any two locations within a vicinity are always aware of each others. Presentities (or actually presentities who take also the role of a watcher) are only aware of each others if they are within the same vicinity. Note that a vicinity need not be *symmetric* meaning that if "presentity p_1 is aware of presentity p_2" this does not imply that "p_2 is aware of p_1". (See further discussion below.)

The *vicinity metric* calculates the distance between locations and specifies whether presentities occupying a location are within the same vicinity or not.

Much like presentities, vicinities provide *presence information*. It means that a vicinity tells which presentities are present there and which state they have. For example a CSCW application provides the information who has opened a shared document (of course this is not the only information that is offered by a CSCW application).

A vicinity is specified by a selected location within it. For example the vicinity of a of a web site could be specified by its root document.

Classification. Vicinities are classified whether they are *static* or *dynamic* [6], whether they are *disjunctive* or not, whether they are *symmetric* or not and finally whether they are *singleton* or not.

Static vs. Dynamic Vicinities. Static vicinities are specified by a fixed set of locations. Dynamic vicinities in contrast can grow and shrink. An example of a static vicinity might be a office which is bordered by its four walls. A vicinity metric could be defined here the following way: The distance between identical locations (maybe two people sitting at the same chair) is zero. The distance between two different locations within the same office equals 1. Finally the distance between two locations which are not within the same office equals ∞. In this case people who are in the same office (not necessarily sitting at the same chair) thus whose distance is lesser than ∞ are within the same vicinity and are aware of each other. This scenario matches our workaday experience.

An example for a dynamic vicinity could be people who are aware of each other while taking a walk. Here the distance could be determined by simply measuring the distance in meters between two people (more formally the distance between the locations where people are at a certain moment in time have to be measured). The difficulty here is to define beyond which distance people are no

longer aware of each other. It depends not only on the pure distance but also on the surroundings. If something is in between the direct way of two people, maybe a building or plants or a knoll, so that they can not see each other then the distance between them can be set to ∞ and they are not aware of each other.

Disjunctive vs. Non Disjunctive Vicinities. If two vicinities v_1 and v_2 are disjunctive a presentity p_1 of v_1 is not aware of a presentity p_2 of v_2 and vice versa and that holds for all $(p_1, p-2)$ of (v_1, v_2). The "office" vicinity mentioned above is an example of disjunctive vicinities: People in different offices typically are unaware of each other, if awareness is in terms of "they can not see each other" (at least if doors are closed). The "take a walk" vicinity is an example of non disjunctive vicinities: A walker Marc sees (or is aware of) a walker Mikhail and Mikhail sees Holger but Holger can't see Marc (and vice versa) because in between them there is a building (see Figure 1). So there are two non disjunctive vicinities: one build up by the current locations of Marc and Mikhail and one build up by the current locations of Mikhail and Holger. The current location of Mikhail is in the intersection of both vicinities.

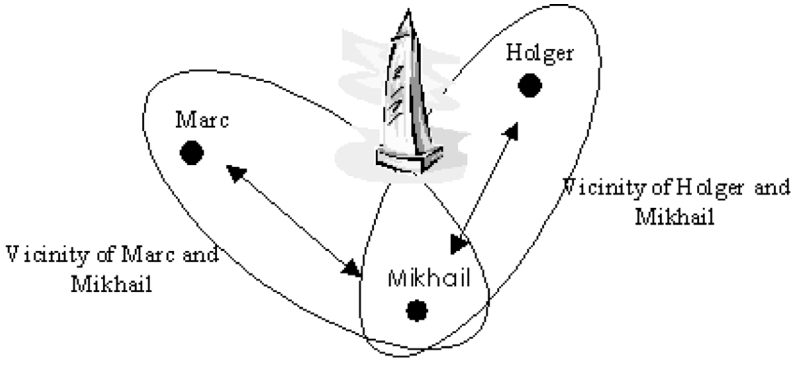


Fig. 1. Who can see whom

Symmetric vs. Non Symmetric Vicinities. A vicinity is symmetric, if the fact that a presentity p_1 is aware of a presentity p_2 implies that p_2 is aware of p_1. The vicinity examples we have given so far are symmetric. Non symmetric vicinities might be such which consist of locations specified by document keywords. Document keywords can be combined by using logical operators or by placing the keywords in quotation marks to denote that a corresponding document contains a exact phrase. Such a combination of keywords is the location specification here.

A vicinity metric could look like this: The distance between documents with the same location specification equals 0. For example the distance between all documents which contains the phrase "little red riding hood" equals 0. The

distance between documents where one location specification "encloses" another equals 1. The distance between documents where one location specification "forecloses" another equals ∞. To continue in our example the distance between all documents which contains the keywords "little" and "red" and "riding" and "hood" and all documents which contains the phrase "little red riding hood and the big bad wolf" equals 1 but vice versa the distance equals ∞. That is because documents which contain the phrase "little red riding hood and the big bad wolf" surely also contain the keywords "little" and "red" and "riding" and "hood". But on the opposite documents which contain the keywords "little" and "red" and "riding" and "hood" need not to contain the phrase "little red riding hood and the big bad wolf". To make things more descriptive, people who have opened or who are interested in documents which contain the keywords "little" and "red" and "riding" and "hood" are also interested or want to be aware of people who are interested in documents which contain the phrase "little red riding hood and the big bad wolf". But not so vice versa because maybe "the big bad wolf" doesn't appear in documents containing the keywords "little" and "red" and "riding" and "hood" or the keywords "little" and "red" and "riding" and "hood" have nothing to do with the well known fairytale characters.

Singleton vs. Non Singleton Vicinities. Finally a vicinity may be categorized whether it is *singleton* or not. A singleton vicinity consists of only a single location whereas non singleton vicinities consists of multiple locations. This distinction makes only sense in case of static vicinities because dynamic vicinities can crow and shrink as described above. Singleton vicinities have relevance in practice: In a CSCW application a user often wants to know what happens at only a single shared resource (e.g. a document or a single paragraph). Therefore she or he subscribes to a vicinity which contains only that document.

3 Kinds of Presence Awareness

Based on the entities discussed so far, we will now describe the different types of presence which can be modelled by using them.

3.1 Presentity Based Presence Awareness

This kind of presence awareness lets a watcher recognize if a presentity is currently present or not and if so at which location. Additional state information may be provided. Informations like history of visited locations or forecasts where a presentity will be in future, in our opinion doesn't pertain presence awareness because, as this phrase implies, it is about the presence not the past or the future. The negation of the question if a presentity is present conflicts with the definition of a presentity given above: If a presentity isn't present it can not be at a location and can therefore not be a presentity. *Persistent presence* means that a presentity is always present even it is not available. (maybe on vacations or sleeping or shut down or whatever) In case of the "not being present" state of

a presentity its *substitute* persistently provides presence and property information. Presence information in this case could be "I am nowhere and my state is unavailable". Another possibility is to provide simply no answer to the presence information question. No answer could be interpreted again as "I am nowhere and my state is unavailable".

Current examples where such kind of presence awareness is partially used are the buddy lists of messengers like from AOL, Microsoft or Yahoo.

3.2 Vicinity Based Presence Awareness

This kind of presence awareness provides information which presentites are within a certain vicinity as well as their exact locations. State information on each presentity is provided.

To revert to the "office" example above, a watcher (or someone who is looking into the office) might get the information that e.g. a colleague is sitting at his desk and another colleague is standing in front. The state of both could be "talking". A collaborative browsing environment could provide the information who is browsing at a certain web site. Here the web site is the vicinity. The exact web page which each user or presentity has opened as well as his state (e.g. "ready to chat") is also provided. Note that a watcher of a vicinity need not be a presentity of that vicinity.

In contrast to presentity based presence awareness, vicinity based presence awareness enables meetings and encounters. While sitting in a pub in the real world one encounters close friends, buddies but also complete strangers. In a noffice building one meets colleagues. Maybe somebody is browsing a web site advertising a tourist country. It would be very helpful to talk with cobrowsers about that country to get more detailed information about it. Those people who are around are surely exactly the right ones to answer questions.

4 Presence Awareness Services

Typically presence awareness services provide presence information to watchers. To be able to provide presence information a presence awareness service must obtain presence information from presentities and/or from vicinities. Because there are two entities who provide presence information (presentities and vicinities) and two kinds of presence awareness (presentity and vicinity based presence awareness) there are also two kinds of presence awareness services. These are explained in detail in the following.

4.1 Presentity Based Presence Awareness Service

The information this kind of service provides is described in Section 3.1. This presence service maintains information where a presentity is currently residing. The presence information is posted to the service by presentities. Watchers can

fetch or poll presence information. If they have subscribed to presence information they get a notification each time presence information changes, thus each time the location or state of a Presentity changes. Figure 2 illustrates the flow of information.

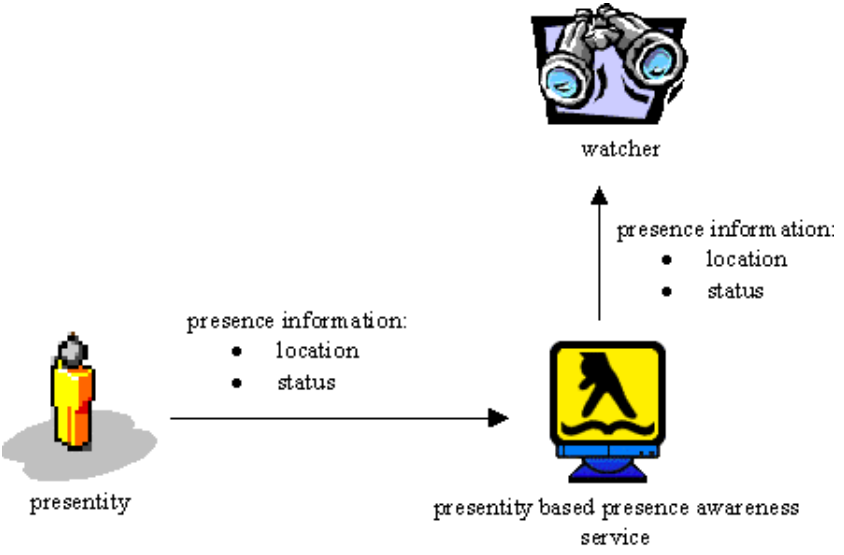


Fig. 2. Flow of information in a presentity based presence awareness service

4.2 Vicinity Based Presence Awareness Service

The information this kind of service provides is described in Section 3.2. This service maintains information which presentities are currently within a certain vicinity. Conceptualy the presence information is posted to the service by the vicinities. (in a real implementation the service could group locations into vicinities respective to a certain location specification and the vicinity metric used.) Watcher can again fetch or poll for presence information. If they have subscribed to presence information they get a notification each time a presentity enters or leaves a vicinity or each time a presentity of that vicinity changes its state. Figure 3 illustrates the flow of information.

5 Current Work

We have prototyped a presence awareness service which embodies the entities described in the previous sections of this paper. (especially the presentity resp. vicinity based presence awareness) It has a client/server architecture with a lean

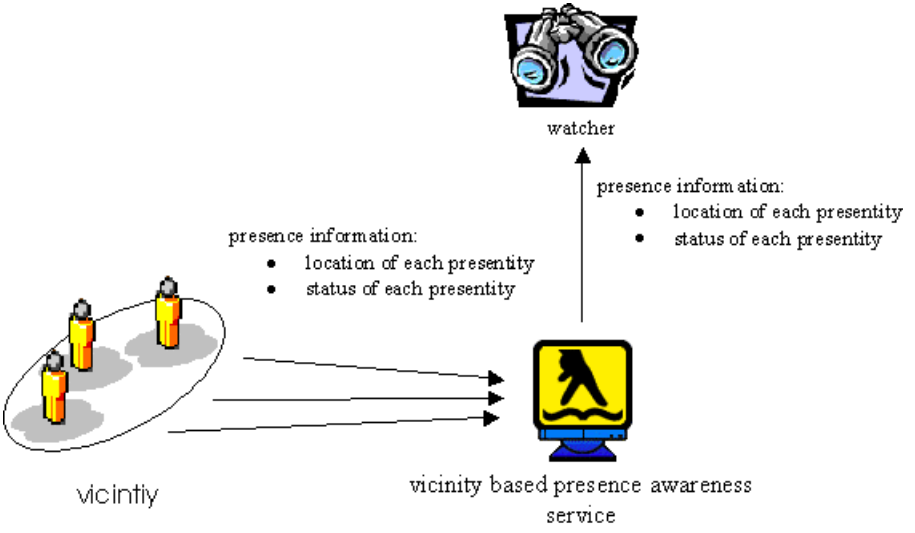


Fig. 3. Flow of information in a vicinity based presence awareness service

client side API. We use Java as a programming language to ensure that our service can be installed on every platform that provides a Java virtual machine. Our service deals with various underlying database systems like SQL databases, LDAP directory services or proprietary solutions and with arbitrary database schemes to store the persistent properties of a presentity. Furthermore the service uses the standard Java event listener mechanism to notify watchers about changes of presence information or properties.

For those clients who do not want to use the client side Java API we will offer soon a variant of the SIMPLE [8] protocol which will be spoken by the server side of the service. SIP [9] (Session Initiation Protocol) is an IETF standard signaling protocol for initiating interactive user sessions that involves multimedia elements such as voice, video or simply chat. SIMPLE (SIP for Instant Messaging and Presence Leveraging) is an extension of SIP which adds instant messaging and presence support.

Currently we develop an Java applets based collaborative browsing environment based on our service. Further it is a part of a CSCW application likewise currently implemented within the application research program of the GigaPort [10] project.

6 Security and Privacy

To ensure privacy as well as security in terms of authentication and encryption is of high importance for any service offering presence awareness. Few people will

tolerate an observer while doing e.g. home banking. An employee only wants to be seen by his colleagues but typically not by the rest of the world. Publishing a phone number to everybody may also not be wished by some. Especially with aid of services which implement presentity based presence awareness it is very easy to record the visited locations of a presentity. This would violate privacy rights of users exceedingly.

Privacy settings specify access rights of a watcher to presence information as well as on properties (normally only read permission). Role-Based Access Control [11] means that access rights are assigned to roles instead of to single watchers. A watcher who takes a role, is assigned the access rights of that role. Doing so eases maintenance of access rights because roles are much more static than single users (a secretary may dismiss but her role "secretary" remains) .

An example of use could be the seven dwarfs creating a group "friends-of-us" and specifying that members of this group are allowed to read all their presence and property information. Next they assign only snow-white to this group. The result is that only snow-white knows where the dwarfs are and above all the jealous witch will never know when the dwarfs are not at home.

Privacy settings may transform an actual symmetric vicinity to a non symmetric one because a presentity with stronger privacy settings is able to see other presentities with weaker settings but not vice versa.

7 Future Work

Security and privacy is of a high importance. We will add security and privacy components to our service.

Currently the only kind of locations our service handles are URLs. We intend to add locations specified by GPS coordinates, and locations specified by document keywords together with corresponding vicinity metrics.

Another very important part of our future work is to implement a distributed variant of our service to make it scalable. Currently we have only a single server to which all users are connected. It means if two servers are running and two users are at the same location but logged on at different servers they are not aware of each other because currently the servers do not communicate.

It is important to note that making our service distributed is not only a matter of communication but also of distributed vicinity calculation. Each process needs to know which part of a distributed vicinity is in its scope and which other processes are involved in vicinity calculation and needs to be notified if something has changed in its part.

References

1. Siemens Location Services – mobile phones with a sense of place added value through location services; White Paper
2. Collaborative Browsing in the World Wide Web Gabriel Sidler, ETH Zurich; Andrew Scott, Lancaster University; Heiner Wolf, University of Ulm; Proceedings of the 8th Joint European Networking Conference, Edinburgh, May 12.–15. 1997.

3. ShopAware Project on the Web.
http://ernst.informatik.uni-ulm.de:81/shopaware/
4. RFC 2778 “A Model for Presence and Instant Messaging”.
http://www.ietf.org/rfc/rfc2778.txt
5. Presence: The Best Thing That Ever Happened To Voice; Jonathan Rosenberg, dynamicsoft. http://www.cconvergence.com/article/CTM20001023S0001
6. User Space Meets Document Space Konrad Froitzheim, Heiner Wolf, University of Ulm; to appear in the Shortpaper Proceedings of WWW7, Brisbane, Australia, April 1998.
http://www-vs.informatik.uni-ulm.de/Papers/www7-short/461.html
7. Java Tutorial from Sun Microsystems on the Web.
http://java.sun.com/docs/books/tutorial/
8. SIMPLE on the Web.
http://www.ietf.org/html.charters/simple-charter.html
9. SIP on the Web. http://www.ietf.org/html.charters/sip-charter.html
10. GigaPort Project on the Web. http://www.gigaport.nl
11. Role Based Access Control David Ferraiolo, Richard Kuhn; National Institute of Standards and Technology Gaithersburg, Maryland; Proceedings of the 15th National Computer Security Conference, 1992

SecAdvise: A Security Mechanism Advisor

Rima Saliba[1], Gilbert Babin[2], and Peter Kropf[1]

[1] University of Montreal
DIRO, Montreal, Quebec, Canada
{salibar, kropf}@iro.umontreal.ca
[2] HEC – Montreal
Information Technology, Montreal, Quebec, Canada
Gilbert.Babin@hec.ca

Abstract. The proliferation of incompatible e-commerce systems applying different security technologies imposes difficult choices on all the concerned parties. In this context, the purpose of this research is to provide the necessary background to develop a security advisor (SecAdvise), which will make it possible to integrate the security mechanisms and the dynamic selection of the various mechanisms between several parties wishing to conduct business transactions safely. Such an advisor aims multiple goals: overcoming compatibility and interoperability problems, evaluating and reducing technological security risks, and enhancing trust.

1 Introduction

The Internet is becoming an increasingly important channel for e-commerce where complex business interactions involve multiple parties. Clearly, the safety of the transactions using electronic means is of capital importance. Several security systems have been implemented [1,4,9] and are operational in many e-commerce applications. The mechanisms employed, the security services, the cryptographic algorithms, the amount of money involved in a transaction, the parties concerned, etc, distinguish these security systems. In this context, the purpose of this research project is to analyze the various types of security threats, mechanisms and services in e-commerce applications, to evaluate them, qualify them, and develop sound methods to select the most appropriate and effective mechanisms and services in a given business and technology context. The results of these investigations will provide the necessary background to develop a security advisor, which will make it possible to integrate the mechanisms and the dynamic selection of the various mechanisms between several parties wishing to conduct business transactions safely. Such an advisor aims multiple goals: overcoming compatibility and interoperability problems, reducing technological security risks, and enhancing trust. The rest of the paper is structured as follows. In Section 2, an introduction to security threats, services and mechanisms is given. This analysis should provide us with a basic understanding of the relationship between risks, services, and mechanisms. We also try to understand

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 35–41, 2002.

the link between security services, mechanisms and the layers of the OSI (Open Systems Interconnection) reference model [2]. Section 3 describes a trust management model developed in [8]. Our advisor is introduced in Section 4. Finally, Section 5 contains some conclusions and future research directions.

2 Security Issues

In [5], Vesna Hassler has identified three principal issues of security: security threats, security services, and security mechanisms. Attacks on systems can be classified in several types:

- *Eavesdropping.* Intercepting and reading messages intended for other principles.
- *Masquerading.* Sending/receiving messages using another principal's identity.
- *Message tampering.* Intercepting and altering messages intended for other principals.
- *Replaying.* Using previously sent messages to gain another principal's privileges.
- *Infiltration.* Abusing a principal's authority in order to run hostile or malicious programs.
- *Traffic analysis.* Observing the traffic to/from a principal.
- *Denial-of-service.* Preventing authorized principals from accessing various resources.

This classification leads to a thorough analysis of the most probable threats and of the system's vulnerabilities to these threats. On the basis of the risk analysis results, we can define a security policy that clearly specifies what must be secured. The functions that enforce the security policy are referred to as security services.

We mention the basic security services defined by the International Organization for Standardization. Whenever possible, we also identify in which layer of the OSI reference model [2] these services may be applied. This relationship between services and OSI reference model layers has already been established in [5], on which we base this classification.

- *Authentication.* Different authentication services are available. Peer entity authentication ensures that a communicating party is really what he claims to be (network layer). Data origin authentication delivers proofs that a piece of information originates from a certain source (network layer).
- *Access control.* Access control ensures that only authorized principals can gain access to protected resources.
- *Data confidentiality.* Data confidentiality can be of different types. To ensure confidentiality between two communicating parties that establish a communication channel, a connection confidentiality service is employed (physical layer). If the communication channel is only logical, the service is referred to as connectionless confidentiality (data link layer). If only certain parts

of messages to be exchanged must be protected, a selective field confidentiality service is used (application layer). Traffic flow confidentiality protects against traffic analysis (physical layer).
- *Data integrity.* Similar to data confidentiality services, data integrity services are different for connection-oriented and connectionless protocols. For connection oriented protocols they may provide message recovery (transport layer). Data integrity services can also protect selected fields of messages only.
- *Non-repudiation.* According to the ISO, non-repudiation services can prevent denial-of-origin of data or guaranty the delivery of data. There are two additional possibilities: non-repudiation of submission and non-repudiation of receipt (application layer).

Security services are built using combinations of security mechanisms. These mechanisms are in turn realized by cryptographic algorithms and secure protocols. Here is a summary of the security mechanisms available, as described in [5].

- *Encryption.* Encryption mechanisms protect the confidentiality of data. We distinguish two types of mechanisms: symmetric mechanisms (e.g., Data Encryption Standard – DES; Advanced Encryption Standard – AES) and public key mechanisms (e.g., RSA).
- *Digital signature.* A digital signature can be generated by a special digital signature mechanism as well as by a combination of encryption mechanisms.
- *Authentication exchange.* Authentication can be based on an encryption mechanism, symmetric or public. Therefore, several mechanisms have been developed whose only purpose is authentication exchange (for example, zero-knowledge protocols and Kerberos, a key distribution system).
- *Access control.* Access control mechanisms are closely related to authentication. They deal with controlling access of the subjects to the divers resources.
- *Data integrity.* Data integrity mechanisms protect data from unauthorized modification. One way to protect data integrity is to use an encryption mechanism. In this way, data integrity and data confidentiality are ensured. Another way to ensure integrity is to use a digital signature mechanism. In this case, integrity and non-repudiation are ensured. If integrity is required without confidentiality or non-repudiation, message digests computed by a cryptographic hash function can be used (e.g., SHA-1, MD5). The message authentication code (MAC) can ensure authentication and data integrity.
- *Traffic padding.* Traffic padding mechanisms keep traffic approximately constant, so that no one can gain information by observing it.
- *Routing control.* Routing control mechanism makes it possible to choose a specific path for sending data through a network, hence avoiding undesirable nodes.
- *Notarization.* Notarization mechanisms are provided by a third party notary that must be trusted by all the participants.
- *Key management.* For the public key encryption, key management and certification authorities are a must.

As is the case with security services, security mechanisms may also be used at different layers of the OSI reference model. To illustrate this, we give a short list of well-known security mechanisms and the corresponding layer to which they apply.

- Application layer: S/MIME, S-HTTP, Secure TELNET.
- Presentation layer: Secure RPC, SASL, SSH.
- Transport layer: SSL, TLS.
- Network layer: IP AH, IP ESP.
- Data link layer: Link encryption, MAC address filtering.

The list of services and mechanisms above is not exhautive. In fact, specific contexts require the development of specialized security services. To illustrate this point, we present here some security services developed for electronic payment. Note that most of these services are provided at the OSI reference model application layer.

- *User anonymity and location untraceability.* These services guaranty that althought the merchant received payment for the goods sold, he cannot identify the buyer. These services may be provided, for example, by chains of mixes [3] and blind signature mechanisms. Another form of this service is the payer anonymity service provided in FV (First Virtual) [7].
- *Non-repudiation of payment transactions.* This service ensures that a payer cannot deny having made de the payment. An example of this service is found in the 3KP payment protocol, using digital signatures [9].
- *Confidentiality of payment transaction.* This service prevents eavesdropping on payment data. This type of service is provided by SET (Secure Electronic Transaction) citeSher00a.
- *Freshness of payment transaction.* This service prevents replay attacks on payment transactions. One approach to provide this service is the use of time stamps such as in the 1KP model [9].

When the payment instrument is digital money, the list of services includes protection against double spending, protection against forging of coins, and protection against steeling of coins. When electronic checks are the payment instrument, other types of services are necessary. For instance, payment authorization transfer (proxy) makes it possible to transfer a payment authorization from one authorized principal to another.

3 A Trust Management Model

Several initiatives, such as Semper [6], tried to have the various electronic payment systems converge in order to work out a common operating platform and assure interoperability between them. Robles *et al.* [8] propose a trust model for agent-oriented electronic business applications. This trust model outlines a methodology to define trust requirements and to associate safeguards with them to increase the protection and trust of electronic business frameworks. It suggests

the definition of a trust problem space (TPS) as a set of all possible situations in the system, in which the e-commerce agents can have trust problems about each other or about the environment (a set of the threats and risks mentioned in Section 2). This space includes various types of attacks, and vulnerabilities due to cheating or misuse of system resources. This TPS is related to a collection of interrelated mechanisms, trust units (TU), to provide safeguards to protect systems and sub-systems, and to increase the trust in the systems or subsystems. A TU is a trust logical unit representing a partial or complete solution or countermeasure to any of those problem subspaces presented in the definition of the trust problem space. It may involve cryptographic protocols (RSA, DSE), control mechanisms or infrastructures.

4 Secadvise Definitions

We intend to develop an optimization model, enabled by a security advisor, to

1. identify and specify the Trust Problem Space (TPS) and the Trust Units (TU) available in the context of a communication to be secured, and
2. make the optimal association between TPS and TU to actually secure that communication.

We provide here a preliminary version of the optimization model:

c transaction to secure. It is the business context/transaction that is performed and that need security.

$\mathbf{U}$ the set of all trust units (TU). A trust unit may be a security mechanism, a security protocol or a security infrastructure.

u a trust unit ($u \in \mathbf{U}$).

$\mathbf{R}$ the set of all non-decomposable security risks, such that $\forall r \in \mathbf{R}, \forall u \in \mathbf{U}$, either u covers r entirely or u does not cover r at all.

r a non-decomposable security risk ($r \in \mathbf{R}$).

$\mathbf{P}$ the set of all potential participants in secured communications.

p a participant ($p \in \mathbf{P}$).

R_u the set of security risks covered by trust unit u ($R_u \in \mathcal{P}(\mathbf{R})$).

R_c the set of security risks that need to be covered in transaction/context c ($R_c \in \mathcal{P}(\mathbf{R})$). These are the Trust Problem Spaces (TPS) defined above.

P_c the set of all participants directly communicating in transaction/context c ($P_c \in \mathcal{P}(\mathbf{P})$), e.g., host A wants to communicate with hosts B.

$A_{u,p}$ the set of participants, trusted by participant p, that can act as third party authority in conducting trust unit u ($u \in \mathbf{U}$, $p \in \mathbf{P}$, $A_{u,p} \in \mathcal{P}(\mathbf{P})$), e.g., the set of certification authorities trusted by a participant. If the trust unit does not require such trusted third party, $A_{u,p} = \mathbf{P}$ to simplify the matching process between trust units.

U_p the set of trust units available to a participant $p \in \mathbf{P}$ ($U_p \in \mathcal{P}(\mathbf{U})$).

$\bar{U}_P$ the set of trust units available to all participants $\forall p \in \mathbf{P}$ ($\bar{U}_P \in \mathcal{P}(\mathbf{U})$)

$$\bar{U}_P = \{u \in \bigcap_{p \in P} U_p \quad | \quad \bigcap_{p \in P} A_{u,p} \neq \emptyset\}$$

$\tilde{U}_c$ the minimal set of units to cover the security risks of transaction/context c ($\tilde{U}_c \in \mathcal{P}(\mathbf{U})$)

$$\tilde{U}_c = U \in \mathcal{P}(\bar{U}_{P_c}) \quad | \quad R_c \subseteq \bigcup_{u \in U} R_u \wedge ||U|| = \min_{U' \in \mathcal{P}(\bar{U}_{P_c})} ||U'||.$$

The set $\mathbf{R}$ is defined as the set of non-decomposable risks. Clearly, defining that set is not a simple task. However, we argue that $\mathbf{R}$ may be constructed. Assuming that we find a trust unit u such that risk r is partially covered by u, we can always define risks r' and r'', with $r' \cup r'' = r$ and $r' \cap r'' = \emptyset$, such that r' is covered entirely by u and r'' is not covered at all by u.

This said, in order to avoid such *ad hoc* decompositions, we intend to provide a preliminary multidimensional classification of risks. Two of the dimensions would be the set of risks identified in Section 2 and the seven layers of the OSI reference model [2].

5 Conclusion and Future Work

The proposed model is an interoperable architecture for the various e-commerce security systems. The advisor will choose the best subspace solution for a given security context, depending on the available mechanisms. Regardless of the kind of the mechanisms, the advisor will assess minimum and acceptable security for the transactions between the various parties wishing to conduct safe business transactions. In addition, the advisor should facilitate the assessment of the trustworthiness of security mechanisms and services, providing a systematic evaluation framework based on the multidimensional risk classification. In the foreseeable future, a detailed classification of the mechanisms will be available which will conduct the trustworthiness of these mechanisms. This classification will help to demonstrate the applicability of the approach. Moreover we will choose a number of mechanisms to test the architecture/advisor. We will also define standard scenarios to implement and test the system.

References

1. W3c security. Available from `http://www.w3.org/Security/`
2. Information technology – open system interconnection – basic reference model: The basic model. ISO/IEC Standard 7498-1, International Organisation for Standardization, 1994.
3. D.L. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.
4. W. Ford and M.S. Baum. *Secure Electronic Commerce — Building the Infrastructure for Digital Signatures and Encryption.* Prentice Hall PTR, second edition, 2001.
5. V. Hassler. *Security Fundamentals for E-commerce.* Computer Security Series. Artech House, 2001.
6. Gérard Lacoste, Brigit Pfitzmann, Michael Steiner, and Michael Waidner, editors. *SEMPER — Secure Electronic Marketplace for Europe.* Number 1854 in Lecture Notes in Computer Science. Springer-Verlag, 2000.

7. D. O'Mahony, M. Peirce, and H. Tewari. *Electronic Payment Systems.* Artech House, 1997.
8. S. Robles, S. Poslad, J. Borrell, and J. Bigham. A practical trust model for agent-oriented electronic business applications. In *Proc. of the 4th Int'l Conf. on Electronic Commerce Research (ICECR-4)*, volume 2, pages 397–406, Dallas, Texas, USA, November 2001.
9. M.H. Sherif and A. Serhrouchni. *La monnaie électronique — systèmes de paiement sécurisé.* Eyrolles, 2000.

Research on Reliable Communication in Real-Time Collaborative Designing Systems*

Xueyi Wang, Jiajun Bu, and Chun Chen

Department of Computer Science and Engineering
Zhejiang University, Hangzhou, 310027, China
xueyiwang@yahoo.com, {bjj, chenc}@cs.zju.edu.cn

Abstract. Real-time collaborative designing systems allow users to view and design the same document from geographically distributed sites connected by web. In these systems, each action should be transferred to other sites correctly, timely and orderly. While a lot of researches focus mainly on providing reliable network environment to support collaborative works, we propose a new kind of reliable communication architecture for collaborative group. This architecture is based on UDP protocol for end-to-end transmission, and adopts a new Site-based Reliable Communication Protocol (SRCP) that depends on collaborative site itself to choose reliable communication channel and provide reliable transmission. Each site checks the connection status and chooses reliable path to transfer actions dynamically. Lost actions can be detected and retransferred faster when comparing to other architectures. The protocol and algorithms are given in detail in this paper.

Keywords: CSCW, real-time collaborative design, reliable communication architecture, Site-based Reliable Communication Protocol (SRCP)

1 Introduction

Real-time collaborative designing system is a subclass of Computer-Supported Collaborative Work (CSCW) system. It has many applications such as textile pattern design, architectural design, and collaborative documentation design in order to shorten the time, reduce the cost, and improve the quality. Comparing to other collaborative systems, real-time collaborative designing system has some differences. For example, videoconference system requires transferring video and audio data timely and allows losing some data, and electric commerce system allows losing no data and transfers data only between client and server. Here we give basic characteristics of real-time collaborative designing system.

Reliable. Actions of each site should be transferred to other sites correctly and orderly. Since communication on the web is unreliable, it is required to construct reliable communication architecture.

* This project is supported by National Natural Science Foundation of China.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 42–53, 2002.

Real-time. Response for local actions should be quickly and latency for remote actions should be low. Since latency may cause conflict and ambiguity, system will not work smoothly if the latency for some actions is long.

The goal of this paper is to design reliable communication architecture to promise reliable transmission and reduce communication latency for actions. We regard collaborative sites as a virtual direct-connected network in which each site directly connects with other sites. As long as these sites are connected, actions can be transferred reliably and timely and lost actions can be detected and retransferred quickly.

This architecture has implemented in the CoDesign prototype system. The CoDesign system is a multi-level collaborative graphics designing system [2,3]. The architecture is as follows: Project – Document – Layer – Object. A project has one or more document(s), a document has one or more layer(s), and a layer has one or more object(s). The object is the minimum operable cell. An object has many attributes, like position, color, angle, etc. Actions include creating objects and modifying object attributes.

The rest of this paper is organized as follows. Background and motivation for our architecture are represented in Section 2. The new reliable communication architecture with corresponding protocol and algorithms are given in Section 3. Analysis of our architecture and comparison are discussed in Section 4. Conclusion and further work are given in Section 5.

2 Background and Motivation

2.1 Previous Work

A lot of researches have been done to achieve reliable communication for groupware. These works mainly adopt enhanced communication protocols to provide reliable network environment for exchanging data on the web [4]. These protocols can be classified into two categories: multicast routing protocols and reliable transmission protocols, discussed as follows.

Several multicast routing protocols such as Mbone, which is based on the model established by Deering [5], have been used in the Internet. The multicast routing protocols aim at constructing distribution tree through the network graph. Several algorithms have been proposed to construct multicast trees, such as Core-based tree [6]. But designing an efficient multicast route requires the knowledge of numerous parameters that are not easy to quantify, such as the topology of the network, the dynamics of the group, the location of the group members, and other routing algorithms already used in the network. Also the dynamic behavior of the group is a problem in designing multicast route.

Collaborative designing system requires reliable communication. Several reliable transmission protocols like SRM [7] have been tested on the Mbone. These protocols mainly use a negative acknowledgement mechanism (NASK) with a retransmission request, while point-to-point transport protocols use a positive

acknowledgment mechanism (ACK) to guarantee reliability. In the NASK mechanism, a site transmits packets without waiting for ACK's, and error detection is performed by destinations using the packet's sequence number. The philosophy is to avoid sending state messages (e.g., ACK's) when everything is normal, so as to improve the communication efficiency. When the communication of two sites is delayed or blocked, both sites can transfer actions to each other again only if the communication is recovered. This mechanism may waste communication efficiency when using it in collaborative system, because other sites in the collaborative group may be communicable with these two sites and can be regarded as middle-sites to transfer actions.

2.2 Motivation

Real-time collaborative designing system can work when we combine the two categories of protocols together: multicast routing protocols for real-time transmission, and reliable transmission protocols for reliable transmission. But each site has only fixed communication channels to send and receive actions with other sites in this system. If these channels are broken, it may cost a lot of time to find other communication channels even when this site connects well with some other sites in the group, and the collaborative work will not be smoothly.

We propose a new kind of reliable communication architecture to improve this situation in Section 3. Here we first analysis the requirements for implementing the real-time collaborative designing system.

Small group. A group of collaborative designing system allows about 10 peoples to work together [10].

Reliable communication. Actions should be transferred to other sites without error.

Orderly execution. All actions can be executed orderly through comparing State Vector (SV) [11,12].

Low latency. Collaboration is a real-time and highly reactive multi-user process where users interact with each other's actions and reactions. It has been suggested that good collaboration environments must have an end-to-end delay of no more than 100ms for transferring actions [8]. Other studies have loosened this requirement to 200ms as acceptable delay [9].

As we can see, collaborative designing system allows working in small group, requires low latency and has difficulty in constructing and maintaining multicast communication protocols. Next we propose the reliable communication architecture, which is based on UDP for end-to-end transmission control, to achieve those requirements.

3 Architecture

3.1 Framework of Real-Time Collaborative Designing System

We first propose the framework of real-time collaborative designing system with the reliable communication architecture. This framework consists of four com-

ponents (Figure 1): Communication Module, User Interface, Designing Software and Database. We discuss these components as follows:

Communication module communicates with designing software and other sites' communication modules. It has the following functions:

Broadcast local actions. Local actions are executed and broadcasted to other sites immediately.

Detect and retransfer lost actions. Communication module detects whether actions are lost, and asks other sites for retransferring lost actions.

Check and choose right communication route. Communication module collects the network connection graph, detects network connection status and chooses the fastest path to transfer actions dynamically.

Communicate with designing software. Designing software delivers local actions to communication module for transferring, and communication module delivers the received executable actions to designing software.

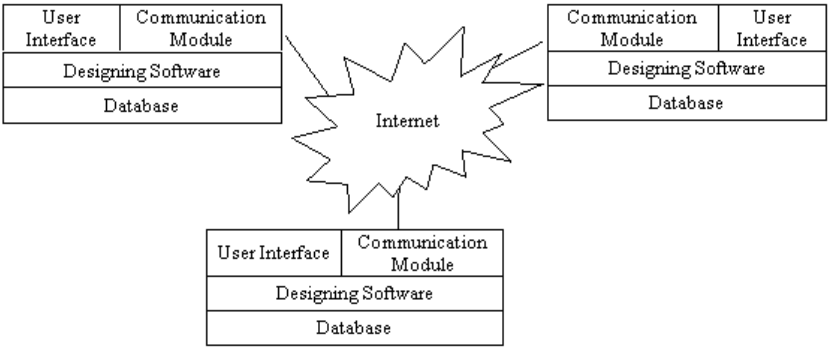


Fig. 1. The framework of collaborative designing system

Database communicates with designing software. It preserves actions for collaborative work and users' information for management. All actions are replicated in each site. After designing software executes an action, the action is preserved in the database.

Designing software communicates with database, user interface and communication module. It manages the local site and has following functions:

Process actions. Designing software executes local and remote actions and disposes conflict situation to make collaborative work smoothly and keep document consistency in all sites.

Manage the database. When joining a collaborative group, designing software reads previous actions from database and asks other sites for new actions. When user generates an action, designing software executes it in local site, stores it into database and asks the communication module for broadcasting.

Communicate with user interface. It accepts user's actions from and delivers the actions and messages to the user interface.

Communicate with communication module. Designing software delivers local actions to communication module for broadcast, and executes remote actions received from communication module.

User interface communicates with designing software and user. It delivers user's actions to designing software and shows actions and messages to user.

These four components are typical for the real-time collaborative designing system and the CoDesign system is constructed by these components. The communication modules of all sites combine the reliable communication architecture. Next we discuss it in detail.

3.2 Reliable Communication Architecture

In the reliable communication architecture, we devise Site-based Reliable Communication Protocol (SRCP), which is based on UDP protocol for end-to-end communication, to achieve reliable and real-time communication for collaborative group. This protocol adopts the receiver-based transmission control approach.

Site-based Reliable Communication Protocol. In this protocol, we use State Vector structure (SV) [11,12] to identify actions, and each site keeps the last executed action's SV. State Vector is a one-dimension array kept in each site, denoted as SV[N] (assume there are N sites in the group, denoted as $0, 1, \ldots, N-1$). Initially, SV[i] = 0, for all $i \in \{0, ..., N-1\}$. If site i generates an action, then site i's SV[i] = SV[i] + 1, and site i broadcast the action with new SV to other sites. If site i receives an action from site j, then site i's SV[j] = SV[j] + 1 and compares it with received action's SV. If site i's SV $\geq$ received action's SV, action is executed; else some actions must be lost and this action is blocked. We can get unique action sequence and find the lost actions by using this structure.

For example, assume there are three sites in the group. If site 1's State Vector SV1 = [2,1,3], it means site 1 has executed one action from site 2, three actions from site 3 and two actions from local site. After site 1 generates a new action, its SV should be [3,1,3]. Site 1 can only execute actions whose SVs are [2,2,3] and [2,1,4] from site 2 and 3, and if site 1 receives an action with SV [2,3,6] from site 3, two action from site 2 and two actions from site3 must be lost.

We keep five arrays for communication at each site in this protocol, as shown in Table 1. The performatives for this protocol are shown in Table 2.

The functions for communication module can be divided into two categories: transferring actions and choosing route. Here we give the algorithms for these functions in details.

Transferring actions algorithm. The actions are divided as local site actions, received actions and lost actions.

Table 1. Arrays for Communication

Array Name	Meaning
ConnectArray[N, N]	Uses to test network connection status with other sites.
ReplaceArray[N]	If local site receives site i's actions from site j, then ReplaceArray[i]=j. ReplaceArray[i]=0 means local site does not yet choose other sites to transfer site i's actions.
BroadcastArray[N,N]	If broadcasting site i's action to site j, then BroadcastArray[i,j]=1.If no broadcasting, BroadcastArray[i,j]=0.
LostActionList	List of lost actions.
ActionBufferList	List of actions that received but cannot be executed immediately.

Table 2. Performatives for Site-based Reliable Communication Protocol

Performative	Meaning	Remark
Action	Sends action to other sites	Followed by action contents, SV and timestamp.
Connect	Tests network connection status.	Send to all other sites. Followed by local site's current SV and connection status with other sites.
RequestTransfer	Asks a site for transferring another site's actions	Followed by another site's identification and local site's recent SV
ReplyTransfer	Agrees to transfer another site's actions	Followed by another site's identification
CancelTransfer	Asks a site for stopping transferring another site's actions	Followed by another site's identification
RequestAction	Asks for lost action	Followed by action's SV
Block	Cannot transfer another site's actions	Followed by another site's identification

1. When generating an action, local site broadcasts Action performative with the action to other sites.
2. When receiving Action or Connect performative, local site checks whether exists lost actions by comparing local site's SV with received one.
   a) If Action performative is received and no actions are lost, site delivers the received action with other executable actions in ActionBufferList to designing software and sends it to other sites according to BroadcastArray. The corresponding items are deleted from LostActionList and ActionBufferList if exists.
   b) If there exists lost action, site records it into LostActionList, sends RequestAction performative to sender, stores the received action into Ac-

tionBufferList, and deletes corresponding item from LostActionList if exists.

3. At each interval, local site sends RequestAction performative for each lost action in LostActionList according to ReplaceArray.
4. Local site discards received action if it is redundant.
5. Local site sends CancelTransfer performative to action sender if the following conditions are satisfied: sender is not the origin site, the corresponding item in ReplaceArray is not zero, and sender is not the corresponding item in ReplaceArray.

For example, suppose there are 3 collaborative sites. For site 1, ReplaceArray is [1,2,3], current SV is [1,1,1], and it receives an action from site 2 with SV [1,4,2], which means one site 1's actions and three site 2's actions and two site 3's actions have been executed before site 2 generates this action. When comparing SV [1,4,2] to site 1's SV [1,1,1], obviously two site 2's actions and one site 3's action are lost. So site 1 will send two RequestAction performative to site 2 with SV [0,2,0], [0,3,0] and one RequestAction performative to site 3 with SV [0,0,2]. Then site 1 records [0,2,0], [0,3,0] and [0,0,2] into LostActionList, and records the action with SV [1,4,2] into ActionBufferList.

It should be noted that we use the receiver-based transmission control approach to achieve reliable communication. Sender never cares for whether the action can be received. Site can find lost actions through comparing SV that contains in Action and Connect performatives.

The selecting routing algorithm has two parts, first we propose a method to achieving sites' connection graph, and then we give the algorithm in detail.

Achieving sites' connection graph method

1. Each row of ConnectArray represents the connection status from a certain site to other sites. The row that represents local site to other sites is modified at local site. Other rows are changed when site receives Connect performative, which contains the communication status from that site to other sites. Local site keeps the latest timestamp received from other sites. If the latest received Connect performative's timestamp is older than the previous one, site will not change the ConnectArray.
2. When site refreshes at regular interval (INV), Connect performative is broadcasted and all cells in local site row are added by an increment (Δ_1). All cells in local site row are subtracted by another increment (Δ_2) when receiving the Connect performative. Note that $\Delta_2 > \Delta_1$, for Connect performative may be lost.
3. Each cell has same minimum value (MIN) and maximum value (MAX). If the value is MIN, the connection status is the best, else if the value is MAX, the connection is blocked.
4. The Connect performative is broadcasted at regular interval (INV). The parameters MAX, MIN, INV, Δ_1 and Δ_2 can be chosen by users to achieve best performance for collaborative work.

For example, we suppose MIN=0, MAX=6, $\Delta_1 = 1$, $\Delta_2 = 2$ and INV=20ms. The time for a communication status from MIN to MAX is 6 intervals (120ms) if connection is blocked, and the time for a communication status from MAX to MIN is 120ms if no Connect performative lost.

Note that we adopt Connect performative to test network connection status. It should be pointed out that only the connection values from local site to other sites are reliable, other connection values are unreliable because Connect performative may also be delayed. We overcome this defect in the following algorithm.

Selecting Routing Algorithm

1. When the collaborative work begins, all sites receive actions from origin site. And the origin site always broadcast actions to other sites no matter they are connected or not.
2. If sum of the connection values of the path from local site to site i is under a door value (DOOR), local site calculates new minimum path from him to i in ConnectArray. After calculating the new path, the site sends RequestTransfer performative to the first site in the path, sends CancelTransfer performative to former site, sends Block performative to the sites that receive site i's actions from local site, and sets value of the corresponding cell in ReplaceArray as 0.
3. In choosing minimum path, if there exists two or more paths whose values are at close range, local site choose the one that needs the middle-site least.
4. At each interval, if finding value of certain site is 0 in ReplaceArray, local site recalculates the new minimum path from him to that site and sends RequestTransfer performative to the first site in the path.
5. If receiving RequestTransfer performative (asks for transferring site i's actions) and the corresponding value in the ReplaceArray is not 0 (means the local site can transfer site i's actions to sender), the local site sends ReplyTransfer performative to source site, sets the corresponding value in BroadcastArray and calculates the new minimum path if possible.
6. When receiving RequestTransfer performative and the corresponding value in the ReplaceArray is 0, the local site sends Block performative to source site.
7. When receiving ReplyTransfer performative, the local site sets the corresponding value in ReplaceArray.
8. When receiving CancelTransfer performative, the local site removes the corresponding value in BroadcastArray.

It should be noted that the local site does not specify the path when sending the RequestTransfer performative, because connection status except those from local site to other sites may not be the latest. And the next site will choose the right path to obtain the actions. Moreover, the more sites attended in the path, the longer transferring time will be, and generally the transferring path will only need one or two sites as middle-site, so the communication cost will be reduced.

For example, suppose there are 4 sites, ConnectArray in site 1 is [[0,1,5,1], [1,0,2,2], [5,2,0,3], [1,2,3,0]], MIN=0, MAX=6, $\Delta_1 = 1$, $\Delta_2 = 2$, INV=20ms, DOOR=4, and ReplaceArray in site 1 is [1,2,3,4]. Since the connection value from site 1 to site 3 is bigger that DOOR value and the new minimum path is 1-2-3, site 1 sends RequestTransfer performative to site 2 and sets the ReplaceArray[3] to 0. When site 2 receives the RequestTransfer performative, it sends ReplyTransfer performative to site 1 and sets BroadcastArray[3,1] to 1. When site 1 receives ReplyTransfer performative, it sets ReplaceArray[3] to 2. The example is shown as Figure 2.

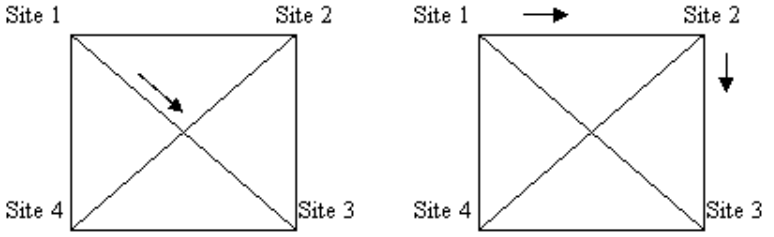


Fig. 2. Example of selecting routing algorithm

4 Comparison to Related Work

Most existing collaborative designing systems have adopted enhanced communication protocols to achieve reliable communication, such as GroupKit [14], CrystalBoard [15], REDUCE [13]. In those systems, reliability is promised by network protocols. The collaborative software needs not care about network communication. If communication is delayed or blocked, it is the router that detects and chooses new transferring path. A typical network structure is Mbone, which is based on UDP for end-to-end transmission control, IGMP for group management, DVMRP for routing. Because of the complexity of the web, designing an efficient multicast route and dynamic behavior of the group will be problems.

Comparing to other multipoint communication architectures, the reliable communication architecture proposed in this paper can provide a reliable and efficient approach for collaborative designing work.

– Collaborative sites are regarded working in a virtual direct-connected network. When the connection of two sites is blocked, these sites can find new path to deliver actions. Here is the reliability comparison of reliable communication architecture and point-to-point communication architecture (Table 3). When using the reliable communication architecture, the reliability is ten times higher than the point-to-point communication architecture.

Table 3. Reliability comparison of point-to-point communication architecture and ours.

Single connection reliability vs. Site number	Reliable communication		Point-to-point communication	
	0.99	0.999	0.99	0.999
3	¿0.999	¿0.9999	0.97	0.997
4	¿0.999	¿0.9999	0.94	0.994
6	¿0.999	¿0.9999	0.86	0.985
10	¿0.999	¿0.9999	0.64	0.956

Note: We assume the connection channel between any two sites is independent from other sites, and the connection channels of the sites in collaborative group form a graph. The point-to-point communication architecture can work only if the graph is complete. The reliable communication architecture can work if the graph is connected. For example, when single connection channel reliability is 0.99, site number is 3, so there are three connection channels. Point-to-point communication architecture can work only if the three connection channels work well, so the reliability is $0.99^3 = 0.9703$. Reliable communication architecture can work if any two connection channels work well, so the reliability is $0.99^3 + 3 \times 0.992 \times 0.01 = 0.9997$.

- Site can detect and receive lost actions faster. In this architecture, local site can know lost actions by checking other sites' Connect performative and receive lost actions from any other site, while the NACK based protocols can only receive lost actions from origin sites.
- Site can also change communication channel faster. As for three basic multicast routing algorithms: the source-based routing algorithm uses flooding approach, the Steiner tree problem is NP-complete and the minimum cost is O(nLogn) (n is the number of nodes in the network), and choosing a center in the center-based tree algorithm is also a NP-complete problem [5]. The maximum cost of the SRCP protocol is O(n2) and the average cost is O(n).
- The latency of transferring actions is low. Since each site broadcasts Connect performative with current SV to all other sites, the time for detecting and receiving lost actions is about the same as INV adds the time for network communication.

The CoDesign system is the first system that adopts this communication architecture to achieve reliable communication. Other works before has never addressed the protocol and algorithms in this paper.

5 Conclusion and Future Work

In this paper, we propose a new kind of reliable communication architecture to provide reliable communication environment for collaborative design. This architecture is based on UDP protocol for end-to-end communication and adopts a

Site-based Reliable Communication Protocol to achieve reliable communication. Major contributions of our work are the reliable communication architecture and the SRCP protocol. Comparing to other communication architectures, this architecture sufficiently takes the advantage the characteristics of real-time collaborative designing system and gets better performance in collaborative designing work.

The SRCP protocol and corresponding algorithms in this architecture have been implemented in the CoDesign prototype system. The CoDesign system is used to test the feasibility of our approach and research for other issues associated with the collaborative graphics design. Our goal is to construct a robust and useful system that can be used in real applications.

There are still other issues for achieving reliable collaborative design, such as version consistency and fault recovery. Now we are working on constructing a complete reliable architecture for real-time collaborative designing work.

References

1. C. Sun, J. Xia, Y. Zhang, Y. Yang, and C. David. : "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing system". ACM Transactions on Computer-human Interaction. 5(1), March 1998, pp. 63–108
2. B. Jiang, C. Chen, J. Bu, "CoDesign – A collaborative pattern design-system based on agent". In: Proceedings of the 6th International Conference on CSCW in Design, Canada, 2001. 319–323
3. Y. Zeng, D. Xu, C. Chen, "Research on Internet based Collaborative Pattern Design System", In: Proceeding of 1999 National Workshop on Computer-Aided Industrial Design and Conceptual Design, 1999, China, 213–318.
4. BEGEL, A. CrystalBoard. CS294-7 "CSCW using CSCW" Course Project Report, December 1997.
5. C. Diot, W. Dabbous, and J. Crowcroft, "Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms". IEEE Journal on Selected Areas in Communications, Vol 15, No. 3, April 1997, pp. 277–290
6. S. Deering, "Multicast routing in a datagram internetwork," Ph.D. dissertation, Stanford Univ., Dec. 1991.
7. T. Ballardie, P. Francis, and J. Crowcroft, "Core based trees (CBT)," in An Architecture for Scalable Inter-Domain Multicast Routing, SIGCOMM '93, San Francisco, Sept. 13–17, 1993, pp. 85–95.
8. S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," in Proc. ACM SIGCOMM '95, Boston, Aug. 28–Sept. 1, 1995, vol. 25, no. 4, pp. 342–356.
9. M.M. Wloka, "Lag in Multiprocessor VR", Presence: Teleoperators and Virtual Environments (MIT Press), Vol. 4, No. 1, Spring 1995.
10. P. K. Shin, and R. V. Kenyon. "Effects of network characteristics on human performance in a collaborative virtual environment". Paper read at Proceedings IEEE Virtual Reality (Cat. No. 99CB36316), 13–17 March 1999, at Houston, TX, USA.
11. Davis, R.C., J.A. Brotherton, J.A. Landay, M.N. Price, and B.N. Schilit, "NotePals: Lightweight Note Taking by the Group, for the Group". Technical Report CSD-98-997, CS Division, EECS Department, UC Berkeley, Berkeley, CA, February 1998.

12. Ellis, C. A. and Gibbs, S. J. “Concurrency control in groupware systems”. In proceedings of the ACM SIGMOD Conference on Management of Data (May). pp. 399–407. 1989.
13. Sun, C., Yang, Y., Zhang, Y., and Chen. D. “Distributed concurrency control in real-time cooperative editing system”. In proceedings of the Asian Computing Science Conference. Lecture Notes in Computer Science, vol. 1179. Springer-Verlag, pp.84-95. Singapore, Dec. 1996.
14. Y. Yang, C. Sun, Y. Zhang and X. Jia. “Real-Time Cooperative Editing on the Internet”. IEEE Internet Computing, May/June 2000 (Vol. 4, No. 3). pp. 18–25.
15. M. Roseman and S. Greenberg, “Building real time groupware with GroupKit, a groupware toolkit,” ACM Transactions on Computer-Human Interaction, vol.3, no.1, pp. 66–106, Mar. 1996.
16. A. Begel and S. Hall. “CrystalBoard: A Transparent Shared Whiteboard”. Final Project Reports CS 294-7: CSCW using CSCW. EECS Department, UC Berkeley, Berkeley, CA, Dec. 1997.

Efficient Connection Management for Web Applications

Yoon-Jung Rhee, Jeong-Beom Kim, Geun-Ho Kim,
Song-Hee Yi, and Tai-Yun Kim

Department of Computer Science & Engineering, Korea University
Anam-dong, Seongbuk-ku, Seoul, Korea
{genuine, qston, root1004, pine, tykim}@netlab.korea.ac.kr

Abstract. HTTP/1.1 may induce wasting server's resource when server maintains connection with the idle-state client that requests no data for a certain time. This paper proposes the mechanism of a connection management supported by the client under persistent HTTP. For the mechanism, we defined finishing time of transmission for HTML page and all embedded file in it as connection-closing time on client-side. For the experimental environment, clients ran on 300Mhz Pentium II PC with 32MB of physical memory running the windows95 and servers ran on OpenWin of Solaris 2.4. An experimental evaluation of connection management policies, conducted using Web server logs, shows that our policy achieves 15-20% reduction on busy Web server in cost with respect to the fixed holding-time policy.

Keywords: Web, TCP, HTTP, Connection Management, Hypermedia Data

1 Introduction

HTTP was designed to be an extremely lightweight stateless protocol on TCP, which is used in World Wide Web distributed hypermedia system to retrieve distributed objects. HTTP messages are transported by TCP connections between clients and servers. Most implementations of HTTP/1.0 [1] use a new TCP connection for each HTTP request/response exchange. Therefore the transmission of a page with HTML content and embedded object files such as image, sound and applet involves many short-lived TCP connections.

TCP connections are established with a 3-way handshake; and typically several additional round trip times (RTT) are needed for TCP to achieve appropriate transmission speed [17]. Each connection establishment induces user-perceived latency and processing overhead. Opening a single connection per request through connection setup and slow-start costs causes problems of performance and latency. Thus, persistent connections were proposed [3,4] and are new a default with the HTTP/1.1 standard [5]. HTTP/1.1 reduces latencies and overhead from closing and re-establishing connections by supporting persistent

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 54–63, 2002.

connections as a default, which encourage multiple transfers of objects over one connection.

HTTP/1.1, however, must decide when to terminate inactive persistent connections. HTTP/1.1 specifies that connections should remain open until explicitly closed, by either party. That is to say HTTP/1.1 does not define explicitly when to terminate TCP connection. Current implementation of HTTP/1.1 uses a certain fixed holding-time model. This model may induce wasting server's resource. Current latency problems are caused by not only network's problem but also server's overloads having limited resource. This paper proposes the mechanism of a connection management on the client-side under persistent HTTP, in addition to HTTP/1.1's fixed holding-time model on server-side. The client exploits the tag information in transferred HTML page so that decides connection-closing time.

This paper is structured as follows. In Section 2 we discuss the related works involved in implementation of persistent connection of HTTP/1.1. Section 3 contains our proposal of connection management. Section 4 reports on experimental results of proposed policy. We finish with a conclusions and future works in Section 5.

2 Issues and Policies of Persistent Connection Management

In this section, we discuss persistent connection management issues involved in using resource. We also describe current implementation policies and its problems of persistent connection management subsequently.

2.1 Persistent Connection of HTTP/1.1

HTTP/1.1 does not specify explicit connection-closing time but provides only one example for a policy, suggesting using a timeout value beyond which an inactive connection should be closed [5]. A connection kept open until the next HTTP request reduces latency and TCP connection.

An open TCP connection with an idle-state client that requests no data consumes a server's resource, a socket and buffer space memory. The minimum size for a socket buffer must exceed the size of the largest TCP packet and many implementations pre-allocate buffers when establishing connections establishment overhead. The number of available sockets is also limited. Many BSD-based operational systems have small default or maximum values for the number of simultaneously-open connections (a typical value of 256) but newer systems are shipped with higher maximum values. Researches indicate that with current implementations, large numbers of (even idle) connections can have a detrimental impact on server's throughput [6,10].

The issues of connection management is to strike a good balance between benefit and cost of maintaining open connections and to enforce some quality of service and fairness issues [10].

2.2 Current Implementation Policies

The current version 1.3 of the Apache HTTP Server [7] uses a fixed holding-time for all connections (the default is set to 15 seconds), and a limit on the maximum allowed number of requests per connection (at most 100). The Apache implementation is a quick answer to the emerging need for connection management. The wide applicability and potential benefit of good connection-management makes it deserving further study.

Persistent connection management is performed at the HTTP-application layer. Current implementations of Web servers use a holding-time model rather than a typical caching model.

Using holding-times, a server sets a holding-time for connection when it is established or when a request arrives. While the holding-time lasts, the connection is available for transporting and servicing incoming HTTP requests. The server resets the holding-time when a new request arrives and closes the connections when the holding-time expires.

In a caching model there is a fixed limit on the number of simultaneously-open connections. Connections remains open "cached" until terminated by client or evicted to accommodate a new connection request.

A holding-time policy is more efficient to deploy due to architectural constraints whereas a cache-replacement policy more naturally adapts to varying server load. Heuristics to adjust the holding-time parameter on server-side were recently proposed and evaluated on server logs [2,9,10]. Demanding additional processing for searching heuristic parameters for every connection, if it was used on popular busy servers, it may have an inferior effect on server performance, so consequently deteriorate overall latencies.

3 Client-Based Connection Management Mechanism

In this section, we propose the mechanism of a connection management supported by client-side under persistent HTTP, in addition to fixed holding-time on server-side. It can reduce properly the numbers of connections with clients on server without increasing latency and improve performance of the Web server by preventing the resources of the server from being wasted.

3.1 Proposal of Connection Management

When a client does a GET on a URL corresponding to an HTML document, the server just sends back the contents of the corresponding file. The client then sends separate requests for each embedded object files such as image, applet, and sound. Typically, however, most or all of the embedded object files reside on the same site as the HTML document, and will ultimately come from the same server.

We define the finishing time of transmission for HTML document and all embedded object file in it as connection-closing time. For this definition to be

implemented, we present a mechanism, which both client and server are able to close the TCP connection. Figure 1 shows time line of proposed connection management policy.

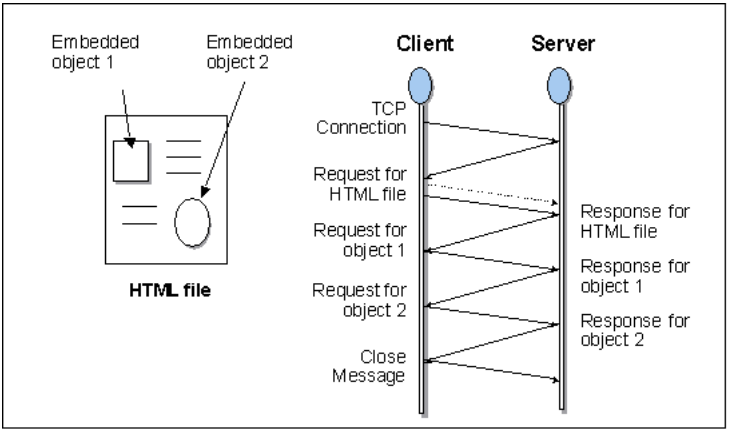


Fig. 1. Operational Time Line of Proposed Mechanism

Following are the mechanism of connection management supported by each client and server.

- Client-side: After first receiving a requested HTML file, client parse the HTML document file to find the URLs of the embedded object files and the file names and request pertinent files to the server subsequently. When Ending last embedded file's reception, client close connection with the server.
- Server-side: Server closes connection with client through the fixed holding-time model that maintains connection for a certain time.

3.2 Prototype Algorithm

We present simple algorithm for implementing prototype for our proposal. Used methods are limited to GET message for file request and CLOSE message for closing connection.

The client starts to establish connection with a corresponding server by user's ask, requesting GET message for first HTML file to the server. After receiving HTML document file, client parses tag attribute information (e.g. img, applet, embed) in it about embedded objects (e.g. image files, java applet class files, sound files) and request corresponding object files to the server through GET message. When the last embedded file arrives, client sends CLOSE message to the server for closing current connection and terminates connection. Server also

closes connection when server finishes the connection. Figure 2 shows the client's algorithm presenting connection management.

Server establishes socket and watches incoming connection request. After Received connection request, server establishes connection with the client and sends repeatedly the file corresponding requested file name. Server send CLOSE message to current client for closing connection by the fixed holding-time model that maintains connection for a certain time and close the connection with the client. After then, server releases resources, socket and socket buffer memory having been assigned to the client. Therefore, the next clients requesting connections to the server are able to receive faster and more fairness service. Figure 3 shows the server's algorithm presenting the proposal connection management.

```
Client()
{
  get  new_URL from user
  open SERVER.SOCKET with new_URL
  send GET with URL.file_name
  read HTML_documents stream from server
  while (HTML_documents) {
    parse TAG_attributes in HTML_documents
    if embedded_object exists
      then add file_names to request_list
  }
  while (request_list) {
    send GET with file_name
    read stream from server
  }
  send CLOSE to server
  close SERVER.SOCKET
}
```

Fig. 2. Client algorithm for proposal connection management

4 Experimental Results

In this section, we report on experiments to measure the effect of the new connection management policy on observed latency.

4.1 Experimental Setup and Results

To validate the proposal we implemented prototype clients and servers. The clients and the servers are implemented in the JAVA programming language.

– Two different kinds of prototype clients:

```
Server()
{
  if accept SYN from client {
    open CLIENT.SOCKET
    start holding_time
    while (holding_time) {
      read stream
      if method in stream is GET {
        get file_name from stream
        if file exist  then response requested_file
        else response "file not found"
      }
      else if method in stream is CLOSE
        then close CLIENT.SOCKET and break
      restart holding_time
    }
    send CLOSE to client
    close CLIENT.SOCKET
  }
}
```

Fig. 3. Server algorithm for proposal connection management

  - One for our client-based connection management (C-CM) policy
  - The other for holding-time policy and HTTP/1.0
- Two kinds of prototype servers:
  - One for HTTP/1.0
  - The other for holding-time policy and C-CM policy

Each client ran on 300Mhz Pentium II PC with 32MB of physical memory running the windows95. Each server ran on OpenWin of Solaris 2.4. All of the servers are implemented by thread-based and event-driven feature and collect CPU and physical memory statistics.

For a busy Web server environment, we assume a process-per-request model, with pools of processes. Our mechanism of the servers limits resource usage of processes by limiting concurrency. This is achieved by imposing an upper bound on the number of processes in the pool [19]. If all processes are busy, additional incoming transactions of new clients are delayed (in the OS) until a process becomes available. We measured network retrieval times, not including the time it took to render images on the display.

In our experiments, we measured the time required to load a document and all of its embedded images on the clients. We created documents with different numbers of embedded images, and with images of 45K bytes sizes. We did these measurements for the case accessed via a 1.544Mbit/sec T1 link. Also, we measured the HTTP throughput and CPU utilization of the servers.

Figure 4 shows that load time depends on the number of images retrieved, using busy Web server environment with the size of processes pool limited to

20 and 40 clients requesting access to the servers. In this case, C-CM policy improves latency by about 15-20% than holding-time policy.

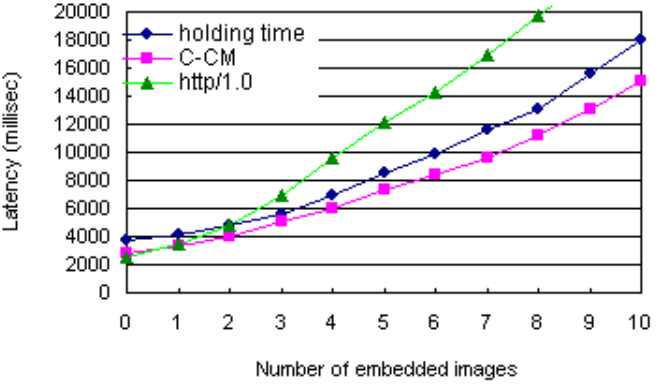


Fig. 4. Latencies for busy Web server environment

In addition, we measured CPU Utilization percentages and latencies according as clients requesting access to servers increase when the size of processes pool was limited to 20 and the number of embedded images in HTML documents was 10.

Figure 5 shows the CPU Utilizations of holding-time policy, C-CM policy and HTTP/1.0. CPU Utilization of C-CM became lower than that of holding-time from 20 that the number of clients is the same size of processes pool.

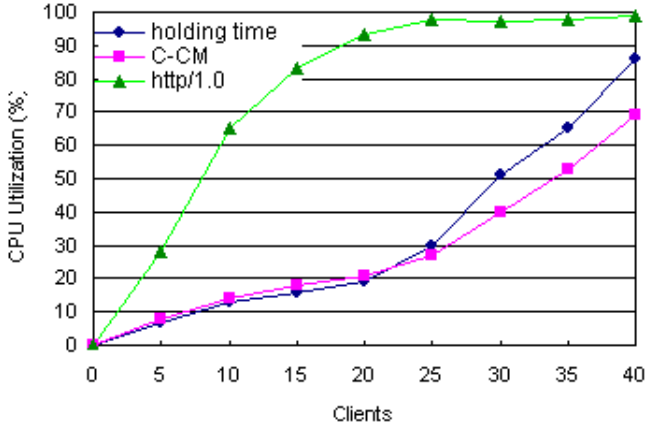


Fig. 5. CPU Utilizations according as clients increase

Figure 6 shows the load time for the holding-time policy, the C-CM policy and HTTP/1.0. Latency of holding-time is slightly lower than C-CM up to 20 clients, but not noticeable. From 20 clients, Latency of C-CM become lower than holding-time.

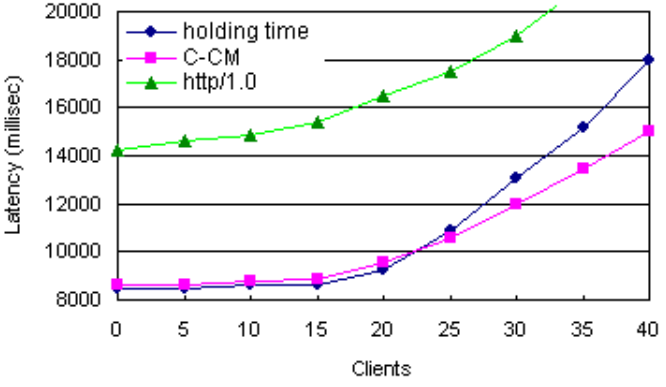


Fig. 6. Latencies according as clients increase

4.2 Discussion of Results

C-CM policy provides significant improvements for busy server transactions. In this case, C-CM policy reduces latency by about 5% to 20% and CPU Utilization by about 5% to 15%.

In HTTP/1.0, each HTTP request/response exchange use a new TCP connection. Therefore the transmission of a page with HTML content and embedded object files involves many short-lived TCP connections. Opening a single connection per request through connection setup increase latency and require more CPU utilization. In holding-time policy, when the number of clients accessing to server is low, the performance in the CPU utilization and latency respect is slight better than C-CM. However, if accesses from clients to a server increase explosively and then concurrent connections of the server with clients are occupied, the access requests of new clients to the server should wait until the holding-time of any idle connections occupied becomes expired and the part of resources such as CPU, memory and socket buffers available.

In C-CM policy, however, they can receive service without waiting for idle connections with connected clients being expired. And advantage of C-CM is no additional overload that is imposed to server and can degrade performance of server if the connection same mechanism runs on server-side, because it runs on client-side. In addition, because C-CM policy needs little extra time for counting the number of embedded objects and gathering the URL of those, there is no additional delay in HTML displaying on client's window.

Our C-CM mechanism was conducted using a small, static set of web pages. Limitation of our study is that the proposed mechanism may have little effect if Web servers should offer dynamic content, containing streaming data such as audio and video that require connection to be kept.

5 Conclusions

We proposed the mechanism of a connection management supported by the client in addition to fixed holding-time model on server-side, under persistent HTTP. For the mechanism, we defined finishing time of transmission for HTML page and all embedded file in it as connection-closing time on client-side. The client exploits the tag information in transferred HTML page so that decides connection-closing time. As the Processing for parsing of tag information in HTML file occurs on client, the mechanism allows server to use server's resource more efficiently without decreasing performance of server-side and give services to clients more fairly. Therefore our mechanism supports a good balance between benefit and cost of maintaining open connections and to enforce some quality of service and fairness issues.

References

1. T. Berners-lee, R. Fielding and H. Frystyk: Hypertext Transfer Protocol – HTTP/1.0 RFC 1945, MIT/LCS, May 1996.
   `http://ds.internic.net/rfc1945.txt`
2. M. Elaud, C.J. Sreenan, P. Ramanathan and P. Agrawal: Use of server load to dynamically select connection-closing time for HTTP/1.1 servers, Submitted for publication, March 1999.
3. J.C. Mogul: The case for persistent-connection HTTP, Comp. Commun. Rev. 25 (4) (1995) 299-313.
   `http://www.research.digital.com/wrl/techreports/abstracts/95.4.html`
4. V.N. Padmanabhan and I.C. Mogul: improving HTTP latency, Comput. Networks ISDN Syst. 28(1/2) (1995) 25–35.
5. T. Berners-Lee, R. Fieding, J. Gettys, J.C. Mogul, H. Frystyk, L. Masinter, and P. Leach: Hypertext Transfer Protocol – HTTP/1.1 RFC2616 Jun 1999.
   `http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf`
6. L.A. Belady: A study of replacement s for virtual storage computers, IBM Systems Journal 5 (1996) 78–101.
7. Apache HTTP server project, `http://www.apache.org`
8. E. Cohen and H. Kaplan: Exploiting regularities in Web traffic patterns for cache replacement, in: Proc. 31st Annu. ACM Symp. On Theory of Computing, ACM, 1999.
9. E. Cohen, H. Kaplan, J. Oldham: Managing TCP connections under persistent HTTP, Elsevier Scince B. V, 1999
10. G. Banga and J. Mogul, Scalable kernel performance for Internet servers under realistic loads, in: Proc. USENIX Annu. Technical Conf., USENIS Assoc., 1998, `http://www.cs.rice.edu/~gaurav/papers/usenix98.ps`

11. R.T. Braden: Requirements of Internet hosts communication layers, FRC 1122, ISI, October 1989.
12. H. F. Nielsen, J. Gettys, A. Baird-smith, E. Prud'hommeaux, H.W. Lie, C. Lilley: Network Performance Effects of HTTP/1.1, CSS1, and PNG, in: Proc. ACM SIGCOMM '97 Conference, Cannes, France, August 1997.
13. V. Jacobson: Congestion avoidance and control, in: Proc. Of the ACM SIGCOMM '88 Conference, August 1988.
14. S. Spero: Analysis of HTTP Performance Problems. `http://sunsite.unc.edu/mdma-releas/http-prob.html`
15. B. Janssen, H. Frystyk and Spreitzer M: HTTP-NG architectural model, `http://info.internet.isi.edu/in-drafts/files/draft-frystyk-httpng-arch-00.txt`, August 1998.
16. W.R. Stevens, TCP/IP Illustrated, Vol. 1, Addison-Wesley, Reading, MA, 1994
17. W.R. Stevens, TCP/IP Illustrated, Vol. 3, Addison-Wesley, Reading, MA, 1994
18. Z. Wang and P. Cao: Persistent connection behavior of popular browsers, `http://www.cs.wisc.edu/cao/papers/persistent-connection.html`
19. L. Eggert, J. Heidemann: Application-Level Differentiated Services for Web Servers, World Wide Web Journal, Volume 3, 133–142.1999 1 Introduction

QoS Performance Improvement for Web Applications

Yoon-Jung Rhee, Jeong-Beom Kim, Geun-Ho Kim,
Song-Hee Yi, and Tai-Yun Kim

Department of Computer Science & Engineering, Korea University
Anam-dong, Seongbuk-ku, Seoul, 136-701, Korea
{genuine, qston, root1004, pine, tykim}@netlab.korea.ac.kr

Abstract. The current Web service model treats all requests equivalently, both while being processed by servers and while being transmitted over the network. For some uses, such as multiple priority schemes, different levels of service are desirable. We propose application-level TCP connection management mechanisms of web server to provide two different levels of Web service, high and low service by setting different timeout for inactive TCP connection. We evaluated the performance of the mechanism under heavy and light loading conditions on Web server. Our experiments show that, though heavy traffic saturates the network, high level class performance is improved by at most 25-28%. Therefore this mechanism can effectively provide QoS services even in the absence of operating system and network support.

Keywords: QoS, Connection Management, Differentiated Service, WWW, Hypermedia Data, HTTP, TCP

1 Introduction

The World-Wide Web is a typical example of a client/server system: in a web transaction, clients send requests to servers, servers process them and send corresponding responses back to the clients. Concurrent transactions with a server compete for resources in the network and server and client end systems. Inside the network, messages contest for network bandwidth and with other messages flowing between the same end system pair and with other traffic present at the time. Inside the end systems, transactions compete for local resources while being processed. Servers implementing the process-per-request (or thread-per-request) model will allocate one process (or thread) to an incoming request.

The current Web Servers suffer from the increasing resource demands due to the explosive growth of the Web [1,3,4,9]. The Web service model treats all transactions equally, according to the Internet best-effort service [6]. Neither the network nor the end systems typically prioritize traffic. However, there are cases where having multiple levels of service would be desirable. Not all transactions are equally important to the clients or to the server, and some applications need to treat them differently. One example is prefetching requests for web pages

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 64–73, 2002.

by proxies; such speculative requests should receive lower priority than user-initiated, non-speculative ones. Another simple example is a web site that wishes to offer better service to paying subscribers.

Ongoing efforts attempt to provide multiple levels of service, both in the server operating system (OS) and in the network. Although promising in the long run, replacing the OS of end systems or upgrading all routers in the network is often impractical. Instead, we will show that substantial benefit can be achieved with server-side, application-level-only mechanisms.

HTTP/1.1 [5] standard reduces latencies and overhead from closing and re-establishing connections by supporting persistent connections as a default, which encourage multiple transfers of objects over one connection. HTTP/1.1 does not specify explicit connection-closing time but provides only one example for a policy, suggesting using a timeout value beyond which an inactive connection should be closed [8,13]. HTTP/1.1 must decide when to terminate inactive persistent connections. Current implementation of HTTP/1.1 uses a certain fixed holding-time model. This model may induce wasting server's resource. Current latency problems are caused by not only network's problem but also server's overloads having limited resource. A connection kept open until the next HTTP request reduces latency and TCP connection.

We propose application-level TCP connection management mechanisms of Web server to provide two different levels of web service, high and default priority by setting different timeout for inactive TCP connection. We evaluated the performance of the mechanism under heavy and light loading conditions on Web server. Our experiments show that, though heavy traffic saturates the network, high level class performance is improved by at most 25-28%. Therefore this mechanism can effectively provide QoS services even in the absence of operating system and network support.

In these two simple examples, external (management) policies control priority assignments. Depending on the nature of the policy, it may or may not be acceptable to delay or drop transactions.

2 Issues of Persistent Connection

The Hypertext Transfer Protocol (HTTP) dominates information exchange over the Internet. HTTP messages are transported by TCP connections between clients and servers. Most implementations of HTTP/1.0 [4] use a new TCP connection for each HTTP request/response exchange. Hence, the transmission of a page with HTML content and embedded images involves many short-lived TCP connections.

TCP connection is established with a 3-way handshake; and typically several additional round trip times (RTT) are needed for TCP to achieve appropriate transmission speed [15]. In addition, Slow-Start mechanism of TCP implementation adds at least one RTT to the total transaction time [12,14]. Each connection establishment induces user-perceived latency and processing overhead. Thus, persistent connections were proposed [9,10,12] and are now a default with

the HTTP/1.1 standard [5]. HTTP/1.1 keeps open and reuses TCP connections to transmit sequences of request/response messages; hence, reducing the number of connection establishments and resulting latency and processing overheads.

Persistent connection management is performed at the HTTP-application layer. Current implementations of Web servers use a holding-time model rather than a typical caching model. Using holding-times, a server sets a holding time for a connection when it is established or when a request arrives. While the holding-time lasts, the connection is available for transporting and servicing incoming HTTP requests. The server resets the holding-time when a new request arrives and closes the connections when the holding-time expires. In a caching model there is a fixed limit on the number of simultaneously-open connections. Connections remains open ("cached") until terminated by client or evicted to accommodate a new connection request. A holding-time policy is more efficient to deploy due to architectural constraints whereas a cache-replacement policy more naturally adapts to varying server load. Policies in the two models are closely related when server load is predictable [7]; a holding-time policy assigning the same value to all current connections is analogous to the cache-replacement policy LRU (evict the connection that was Least Recently Used). In fact, under reasonable assumptions the holding-time value can be adjusted through time as to emulate LRU under a fixed cache size (and hence adapt to varying server load) [7].

We propose different levels of web service based on the holding-time model by setting different timeout for inactive TCP connection to upper and default classes.

3 Designing Connection Management Mechanism for QoS Service

Transactions compete for resources inside the network and at the end systems. Thus, full support for different levels of service for Web transactions would require both network and end system software (OS and applications) to be extended. These extensions are still under development; and even when finished, deployment will take time, because many routers in the network must be updated for the system to be effective. In the meantime, application-level mechanisms promise most of the benefits of an OS/network solution with the additional advantage of being easy to deploy. Only the application software of the server needs to be modified to offer different service levels.

Providing differentiated services would require that the incoming requests be classified into different classes and different levels of service be applied to each class. In this section, we present our differentiated service mechanism, which manage TCP connection by means of offer different levels.

3.1 Proposed Differentiated Service Model

We study the case where there are only two levels of quality of service needed. We classify clients (users) into an upper class (high level of quality of service) for

membership users who pay the fees, and default class (a low level one) for non-membership users who may get the service for free, and assign default holding-time plus additional time to upper class and default holding-time to default class. The upper class users can reduce processing overheads and network latencies caused by closing and re-establishing TCP connection and Slow-Start problem of TCP when holding-time expires, comparing with default class users. Figure 1 shows operational differences according as different holding times.

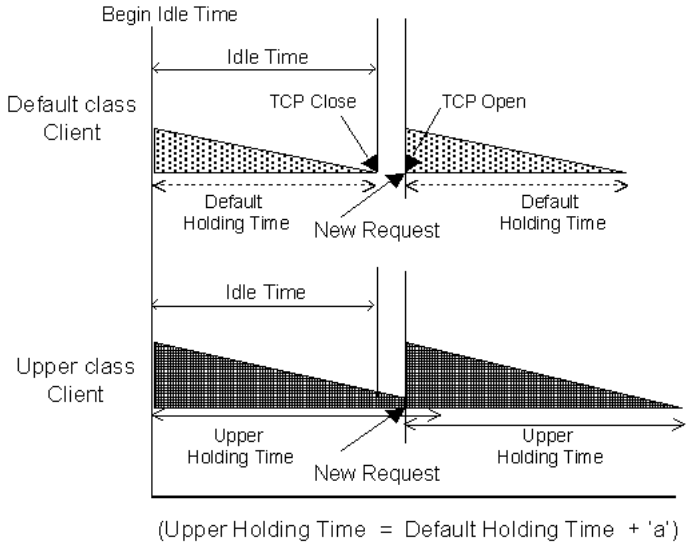


Fig. 1. Operational differences between upper class client and default class client

3.2 Operational Scenario of Proposed System

In this section, we present operational scenarios of proposed system. Then, we show its operational time line of our mechanism in Figure 2.

We suggest *Class Broker* that determines proper holding-time to each client after establishing TCP connection. The Class Broker can be implemented inside the Web server or in membership management DB server. The Followings are roles of Class Broker.

- Manages class table by membership based on membership DB
- Determines proper holding-time of the client when a new client requests access to the Web server

Client starts to establish connection with pertinent server by user's request, and requests HTML file. After receiving connection request from the client, server establishes TCP connection with the client. Then, the server calls the

Class Broker when a user from the client logins to the server. The Class Broker analyzes user's class based on membership DB, and determines proper holding-time of the client and returns the value to the server. The Server assigns the holding-time value returned by Class Broker to the client and sets the holding-time. The server must keep the TCP connection during the holding-time. If receiving new requests from the client, the server resets the holding-time. But, if no request after the elapsed holding-time, the server closes TCP connection with the client and releases resources, socket and socket buffer memory having been assigned to the client.

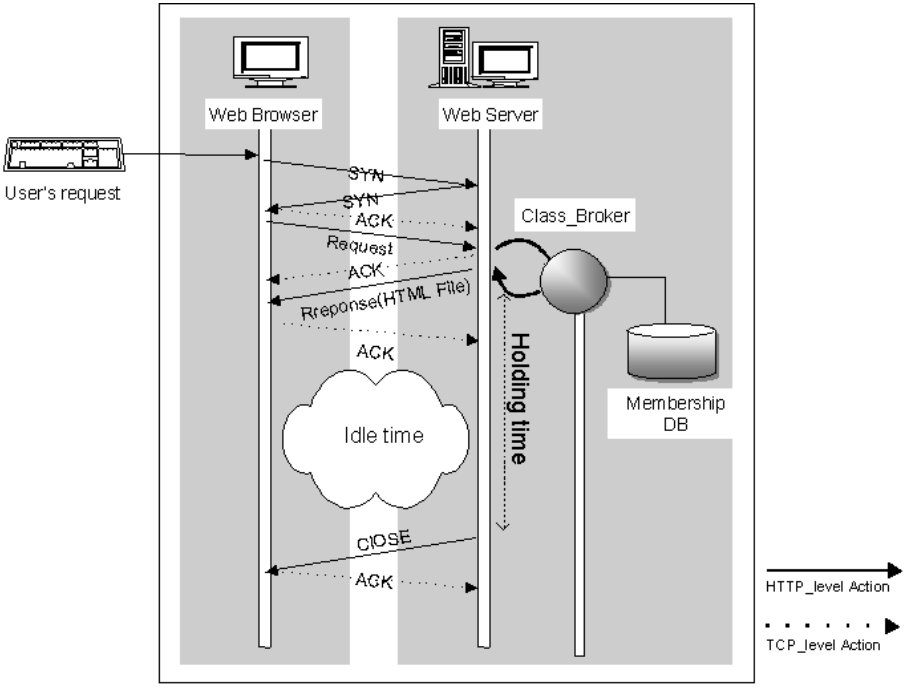


Fig. 2. Time line of proposed mechanism

4 Implementation and Evaluation

If you wish to include color illustrations in the electronic version in place of or in addition to any black and white illustrations in the printed version, please provide the volume editors with the appropriate files.

If you have supplementary material, e.g., executable files, video clips, or audio recordings, on your server, simply send the volume editors a short description of the supplementary material and inform them of the URL at which it can be

found. We will add the description of the supplementary material in the online version of LNCS and create a link to your server.

In order to measure the server latency, we implement a patch file to be linked with Apache code that provides routines for measuring time and logging results. Each process sums the latency for all the requests it handled in each class. Just before finishing its execution, each process writes per-class statistics – average latency and total number of requests – in a log file. Calls to the timing routines were inserted just after a connection is established and just after it is closed, in order to measure the server latency.

4.1 Experimental Setup

Since in current Web servers, there is no difference among requests in terms of different level services. We implemented the QoS connection managements mechanism described above in Apache version 1.3 [2], a popular web server program, modifying by adding a Class Broker, which decides the holding-time in which TCP connection last.

The server machine was a 550Mhz Pentium-III PC with 128MB of physical memory running Linux 2.0.32. In order to obtain effects of simulating busy web server by restricting server's resource, we modified the kernel by increasing the socket listen queue to 30 connections. All of the servers are implemented by thread-based and event-driven feature and collect CPU and physical memory statistics.

The server load was generated by a version of Webstone 2.0.1[16], an industry-standard benchmark for generating HTTP requests. Webstone is a configurable client-server benchmark that uses workload parameters and client processes to generate Web requests. This allows a server to be evaluated in a number of different ways. It makes a number of HTTP GET requests for specific pages on a Web server and measures the server performance, from a client standpoint. In order to generate load for a Web server, client processes request pages after various idle time we set and flies from the server, as fast as the server can answer the requests.

WebStone client processes were spawned in 12 machines (5 client processes per machine) similar to the one used as the server hardware platform. Clients and server communicate through a dedicated 10Mbps Ethernet network. We started WebStone benchmarks, configured to spawn 60 client processes to send requests.

WebStone load is defined by the number of client processes and by the configuration file that specifies the number of pages, their size and access probabilities. In our experiments, we use two different loads, light load causing 20% bottleneck resource utilization and heavy load causing 80% utilization. Heavy loads are representative of the kinds of workload typically found in busy Web Server.

4.2 Results

This section discusses the results obtained for the proposed mechanism. The performance metric is the average latency of a request as perceived by the server. We use 60 client processes and have 30 processes issue requests of upper class and the rest issue requests of default class. We try to analyze the effectiveness of our schemes by comparing the average latency for each class of request with the correspondent latencies when no scheme is used.

In Figures 3 and 4, we set idle time for random value, upper-class holding time for 30 sec and default-class holding time for 15 sec. The graphs show the average latencies when we vary the number of client processes sending requests for the two different loads, light load and heavy load. For larger values of client processes, we note that the performance of upper class is improved by 25% under light load (Figure 3) and 28% under heavy load (Figure 4), while that of default class falls by 3% and 8% under each load (Figures 3, 4), with respect to when no differentiated QoS policy is used. Thus, we can trade good performance for upper class with poor performance for default class.

In Figures 5 and 6, upper-class holding time was set to 10, 20, 30, 40 and 50 sec, and values of idle time is varied from 0 to 60 sec. The two graphs show that if holding time of upper class is shorter than idle time, average latency of upper class is degraded rapidly. The experiment shows that according as holding time increase performance of upper class is improved, because of no need for TCP closing and re-establishing latency overheads. Thus, upper class clients are provided with more reliable service than default class clients, when packets between clients and web server are delayed due to network problems.

4.3 Discussion of Results

In this section, we will summarize the experimental results for our mechanism. An important result of our experiments is that substantial benefits can be provided with user-level changes. Even the very simple approach of assigning more holding time of HTTP/1.1's persistent connection to high level user works well in both case: Web server with light load and one of heavy load.

Our experiments were conducted using a small, static set of web pages. Current Web servers should offer dynamic content, containing large size of multimedia data such as audio, video and image that require reliable service. The server, however, will become difficult to meet the needs within default holding time, when packets between clients and Web server are delayed due to network problems. Our experiments show that proposed mechanisms would be effective in this case by extending open time of TCP connection.

Limitation of our study is that proposed mechanism may degrade server's performance when connections with upper class clients increase rapidly, therefore, all resources get preempted to the existing connections, and requests from new clients would be delayed. To overcome this problem, efficient service scheduling between the existing clients and new clients of upper class needs to be implemented.

Table 1 shows results of comparison between HTTP/1.0, current Web service of HTTP/1.1, upper class and default class of the differentiated QoS mechanism we proposed, in terms of QoS service, minimization of latency, reliable service, and efficient resources management.

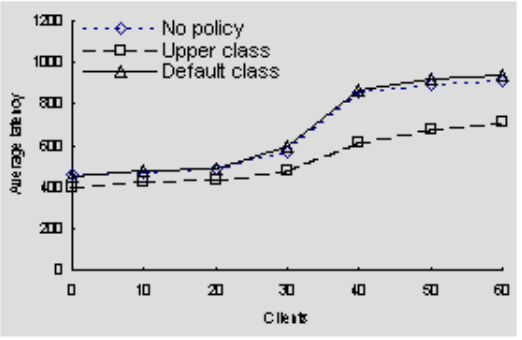


Fig. 3. Average latencies in case of varying the number of client processes (Light load)

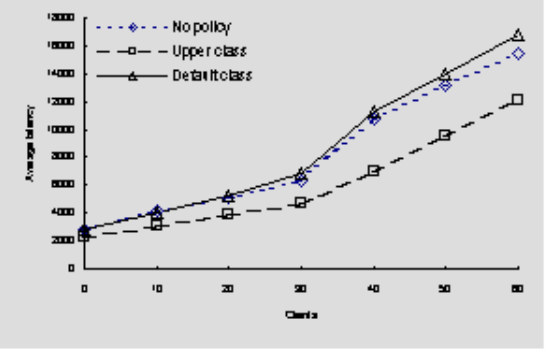


Fig. 4. Average latencies in case of varying the number of client processes (Heavy load)

5 Conclusions

It is impossible for Web server with limited resources to process all of requests from clients by high quality of service when requests increase rapidly. It is an important issue that the Web servers provide different levels of quality of service to members and non-members. We propose application-level TCP connection management mechanisms of Web server to provide two different levels of Web

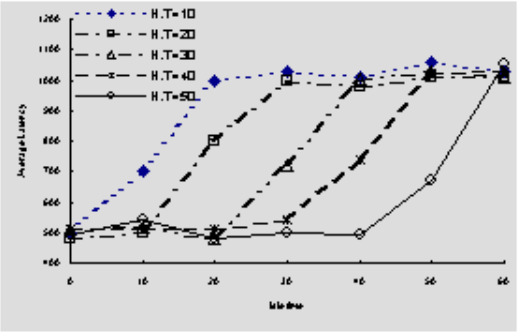


Fig. 5. Average latencies when upper-class holding time set to 10, 20, 30, 40 and 50 sec, values of idle time are varied from 0 to 60 sec (Light load)

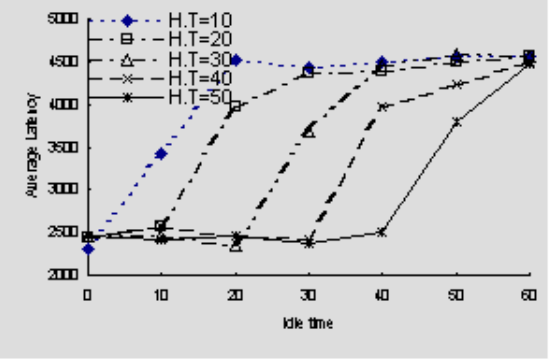


Fig. 6. Average latencies when upper-class holding time set to 10, 20, 30, 40 and 50 sec, values of idle time are varied from 0 to 60 sec (Heavy load)

Table 1. Comparison between HTTP/1.0, HTTP/1.1 and Proposed Mechanism (High: O Low: L None: X).

	HTTP/1.0	Current Web Service (HTTP/1.1)	Default Class	Upper Class
QoS Service	X	X	X	O
Minimization of Latency	X	L	L	O
Reliable Service	X	L	L	O
Efficient Resources Management	O	L	L	L

service, a high and a low service by setting different timeout for inactive TCP connection. We evaluated the performance of the mechanism under heavy and light loading conditions on Web server. Our experiments show that, though heavy traffic saturates the network, high level class performance is improved by at most 25-28%. Therefore this mechanism can effectively provide QoS services even in the absence of operating system and network support.

References

1. Almedia, J., and Dabu, M., and Manikntty, A., and Cao, P., Providing differentiated levels of service in web content hosting, in Proc. 1998 Workshop on Internet Server Performance Madison, Wisconsin, June 23, 1998
2. Apache HTTP Server Project, 1998. http://www.apache.org
3. Banga, G., and Mogul, J., Scalable kernel performance for Internet servers under realistic loads, in: Proc. USENIX Annu. Technical Conf., USENIS Assoc., 1998, http://www.cs.rice.edu/~gaurav/papers/usenix98.ps
4. Berners-Lee, T., Fielding R., and Frystyk, H., Hypertext Transfer Protocol – HTTP/1.0, RFC 1945, MIT/LCS, http://ds.internic.net/rfc/rfc1945.txt (May 1996).
5. Berners-Lee, T., Fieding, R., Gettys, J., Mogul, J.C., Frystyk, H., Masinter, L., and Leach, P., Hypertext Transfer Protocol – HTTP/1.1 RFC2616 Jun 1999. http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf
6. Clark, D., The Design Philosophy of the DARPA Internet Protocols, Computer Communication Review 18, 4, pp. 106–114. 1988
7. Cohen, E. and Kaplan, H., Exploiting regularities in Web traffic patterns for cache replacement, in Proc. 31st Annu. ACM Symp. Theory of Computing, ACM, 1999.
8. Cohen, E. and Kaplan, H. and Oldham, J., Managing TCP Connections under persistent HTTP. 1999 Elservier Science
9. Frystyk, H., Nielsen, J., Gettys, A., Baird-Smith, E., Prud'hommeaux, H.W. Lie and C. Lilley, Network performance effects of HTTP/1.1, CSS1 and PNG, in: Proc. ACM SIGCOMM '97 Conference, Cannes, France, August 1997.
10. Mogul, J.C., The case for persistent-connection HTTP, Comp. Commun. Rev. 25 (4), 1995 299–313. http://www.research.digital.com/wrl/techreports/abstracts/95.4.html
11. Padmanabhan, V. and Katz, R., TCP Fast Start: A Technique for Speeding Up Web Transfers, In Proceedings of the IEEE GLOBE-COM Internet Mini-Conference, pp. 41–46. 1998
12. Padmanabhan, V.N., and Mogul, J.C., Improving HTTP latency, Comput. Networks ISDN Syst. 28(1/2), 25–35. 1995
13. Rhee, Y.-J. and Kim, T.-Y., Heuristic Connection Management for Improving Server-side Performance On the Web, in Proc Workshop on OHS6, Texas, May 30, 2000, LNCS 1903
14. Spero, S., Analysis of HTTP Performance problems, July 1994.
15. Stevens, W.R., TCP/IP Illustrated, Vol. 1, Addison-Wesley, Reading, MA, 1994.
16. Trent, G. and Sage, M., WebSTONE: The First Generation in HTTP Server Benchmarking, Technical Report, MTS, Silicon Graph-ics, Inc., Mountain View, CA, now maintained by Mindcraft, Inc. 1995, http://www.mindcraft.com/webstone/

An Efficient Algorithm for Application-Layer Anycasting

Shui Yu, Wanlei Zhou, Fuchun Huang, and Mingjun Lan

School of Computing and Mathematics
Deakin University, Clayton, Victoria 3168, Australia
{syu, wanlei, fuchun, mlan}@deakin.edu.au

Abstract. Anycasting communication is proposed in IPv6, and it is designed to support server replication by allowing applications to select and communicate with the "best" server, according to some performance or policy criteria, among the replicated servers. Originally anycast researchers focus on network layer. In this paper we pay more attention to application-layer anycasting, because at application layer we can obtain more flexibility and scalability. First of all, we describe the application-layer anycast model, and then summarize the previous work in application-layer anycasting, especially the periodical probing algorithms for updating the database of anycast resolver. After that, we present our algorithm, the requirement-based probing algorithm, an efficient and practical algorithm. In the end, we analyse the algorithms using the queuing theory and the statistics characteristics of Internet traffic. The results show that the requirement-base probing algorithm has better performance not only in the average waiting time for all anycast queries, but also in the average time used for an anycast query.

1 Introduction

An anycast message is the one that should be delivered to one member in a group of designated recipients [12]. Anycasting is an extension of the traditional unicast message, in which there is only one server in the recipient group. With the dramatic development of Internet, more and more applications demand anycast services, for example, when there are a number of mirrored web sites on Internet, user can access the "best" one transparently by anycast service. As the result, in the latest version of IP specification, IPv6, anycasting has been defined as a standard services [5]. Generally speaking, there are two categories of anycasting related problems: procedures and protocols at network layer for routing and addressing anycast messages, and management methodologies at application layer for using anycast services [7].

Some research has been carried out on routing anycast messages [9,10,11,17]. These work are very important for the next generation of IP, and the researchers also obtain some excellent achievements. To the various practical applications of current Internet, however, network layer anycasting has some inevitable disadvantages, which are summed up as following:

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 74–83, 2002.

- Routing of network layer anycasting needs support of routers. The original routers for IPv4 do not support anycasting, therefore we must upgrade routers, and make them recognize anycast addresses and forward packets properly, furthermore, routers must coordinate with each other to complete the delivery of anycast packets correctly. As a result, anycasting services in network layer need a long and expensive transitional period.
- Network layer anycasting is not flexible. Routing of anycast packets are decided by previous fixed routing algorithms entirely within the network, there is no possibility for users or developers to change the algorithms to meet their own special requirements.
- Network layer anycasting can not satisfy the various metrics of Internet applications. The current network layer anycasting algorithms focus on shortest path metric, such as hop count. It is efficient to determine the shortest path, but only for this purpose. It can not handle a variety of other metrics, such as network performance, server throughput, response time, etc.

On the other hand, some research has been carried out on application layer anycasting [1,2,8,15]. Those work try to implement the functionalities of anycasting in application layer, which can avoid the disadvantages outlined previously.

Internet is complicated and dynamic, but it has its own rules actually. There are lots of work have been done about this topic [4,13,14,16]. In this paper, we explore the algorithms on application-layer anycasting from the views of statistics and stochastic.

The rest of the paper is organized as follows. Section 2 discusses the related work on application-layer anycasting, and also the statistics features of Internet. In Section 3, after describing the previous work, we present our novel algorithm on application-layer anycasting. We compare the performance of the application-layer anycasting algorithms in Section 4. Finally, in Section 5, remarks and future work are presented.

2 Related Work and Background

2.1 Characteristics of Internet Traffic

Network traffic properties have been intensely studied for a quite long time. Examples of analysis of typical traffic behaviors can be found in [3,13].

Traffic variables on an uncongested Internet wire exhibit a pervasive nonstationarity. As the rate of new TCP connections increases, arrival processes (packet and connection) tend locally toward Poisson, and time series variables (packet sizes, transferred file sizes, and connection round-trip times) tend locally toward independent [4]. Here the Poisson arrivals are given by

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \quad k = 0, 1, 2, \ldots \tag{1}$$

The analysis later in this paper is based on Poisson arrival, which is described by this formula.

The statistical properties of the Internet congestion reveal long-tailed (log-normal) distribution of latencies [16]. Here latency times T_L are given by

$$P(T_L) = \frac{1}{T_L \sigma \sqrt{2\pi}} \exp\left(-\frac{(\ln T_L)^2}{2\sigma^2}\right) \tag{2}$$

Where σ represents the load of the network. Latencies are measured by performing a series of experiments in which the round-trip times of ping packets are averaged over many sent messages between two given nodes.

2.2 The Application-Layer Anycasting Model

The research of application-layer anycasting is a topic started recently [1,2], but the idea began with the server or resource finding problem about 10 years ago. Initially, with low to moderate server loads, the problem was how to find the desired resource over the network knowing only its name or property. More recently, the Service Location Working Group of the IETF is considering the design of the Service Location Protocol, which allows a user to specify a set of service attributes, which can be bound to a server's network address in a dynamic fashion [2].

With the eruptive development of Internet, users have to face more and more disadvantages of Internet, and now, users have to constantly to find the "best" service from among many content-equivalent servers. There are several outstanding studies in this area: 1) Partridge, Mendez and Milliken [12] proposed the idea of anycasting at the network layer. 2) A study by Guyton and Schwartz [6] which addresses the problem of locating the nearest server. 3) Dong Xuan, Weijia Jia, Wei Zhao and Hongwen Zhu [17] presented a simple and practical routing protocol for anycast message at the network layer. And 4) Ellen W. Zegura and the research group [1,2] analysed the limitations of network layer anycasting and offered an idea about application layer anycasting, which will be discussed more in the rest of this section.

The architecture of application-layer anycasting is shown in Figure 1. A client tries to find a service from the replicated servers on the Internet. First of all, the client sends an anycast query to the anycast resolver to decide which server among the replicated servers is the "best". Then an anycast response is obtained, which consists of the "best" service server's website name or an IP address. The rest of the transaction is the traditional unicast operations.

Anycast resolver is the kernel of the whole architecture, it makes use of the anycast domain names (ADNs). The function of an application-layer anycast service is to map an anycast domain name into one or more (unicast or multicast) IP addresses.

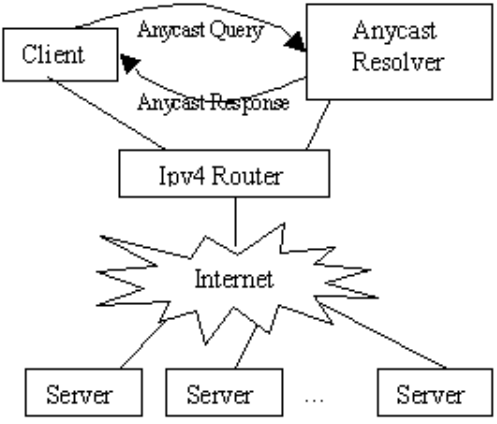


Fig. 1. Architecture of application-layer Anycasting

3 Algorithms for Application-Layer Anycasting

3.1 Periodical Probing Algorithm

The critical problem of application-layer anycasting is how to map an anycast query into one or more IP addresses. [2] presents 4 metrics about how anycasting performs: 1) server response time, 2) server-to-user throughput, 3) server load, and 4) processor load. The paper identified four possible approaches to maintain replicated server performance information in the anycast servers' database:

1. Remote Server Performance Probing: By this methodology, probing agents query the replicated servers periodically to determine the performance that will be experienced if a client were to actually request service.
2. Server Push: The replicated server sends (or pushes) the relevant local performance information onto anycast resolvers.
3. Probing for Locally-Maintained Server Performance: Each replicated server maintains its own locally monitored performance metrics in a globally readable file, remote probing locations can then read the information in the file to obtain the desired message.
4. User Experience: This technique is to collect information about past experience, and then offers a coarse method of maintaining server performance.

As we found that in [2], the foundation of anycast resolver algorithms is the remote server performance probing based on periodical probing, called the periodical probing algorithm. [2] mixed the different methods together in practical applications. There are several disadvantages for periodical probing:

– Accuracy problem. We suppose that the period of probing is ΔT, then during ΔT time, the anycast resolver makes all its decisions based on the result of

last probing. As we know that the Internet changes quickly, therefore the longer the ΔT is, the worse the accuracy.
- Network load problem. In order to improve accuracy, the ΔT should be as short as possible, but, on the other hand, there must get too much probing packets, this will generate a heavy network load.
- Completeness problem. Periodical probing can represent the performance of the servers, but it can not tell resolvers the performance of current network circumstance, which is also an important element for the whole performance.
- Resolver server load problem. The periodical probing algorithm probes for all anycast groups, including the anycast groups which are not used in the coming period. This part of job is not necessary, and it can degrade the resolver's performance.

3.2 Requirement Based Probing Algorithm

In this paper, we present an algorithm, called requirement based probing algorithm, which can overcome all the disadvantages of the periodical probing algorithm, which are mentioned in Section 3.1. The main idea of requirement-based probing algorithm is described below.

When an anycast query is received by an anycast resolver, the resolver will send probing packets, such as ping, to each member in the anycast service group, respectively. In this case, the probed servers should respond for the ping requirements, respectively. If a server's load is heavy or performance is bad, then the respond must last longer than a server whose load is light or performance is good. Therefore the probing packets can not only probe the servers' load or performance at that short period, but also the network load at the same period. Based on the analysis, we define that the first responsive server is the best one among the anycast service group, because the responsive time represents the network performance and server performance as well, then the anycast resolver will submit the IP address of the server to the client via the anycast response. The client then tries to find the server using the traditional IPv4 procedures.

The advantages of our algorithm include higher accuracy, better system performance, and less load for both network and resolvers than the periodical probing algorithm. It is also practical and easy to implement. In Section 4, we will present the performance comparison of the application-layer anycast algorithms.

4 Performance Comparison of Anycast Algorithms

In this section, we compare the two algorithms based on previous research on statistics characteristics of Internet traffic and queuing theory. There are some assumptions for the calculations:

1. Customer arrivals are Poisson arrival.
2. The time unit for both algorithms is 1.
3. During the time unit of 1, there are N customers for both algorithms.

4. There is one server in the system acting as the resolver, and the service velocity, μ, can be obtained from Equation 3.

$$\mu = \frac{1}{E(x)} = e^{\frac{\sigma^2}{2}} \tag{3}$$

There are two important parameters to measure the performance of a system. One is the average time used in the system for a customer, denoted as T_q. Another one is the average waiting time for all customers, denoted as T_w. For both algorithms, we will calculate these two parameters respectively.

4.1 System Performance of the Periodical Probing Algorithm

For this algorithm, we make the following two reasonable assumptions:

1. There are two segments in one period, p and $1-p$, shown in Figure 2. During time points 0 and p, there is no customer arrival; during time points p and 1, there are N customer arrivals, and the rule is Poisson arrival.
2. During time points 0 and p, anycast resolver provides service to client, and during time points p to 1, there is no service for clients. In this duration, the anycast resovler updates its database.

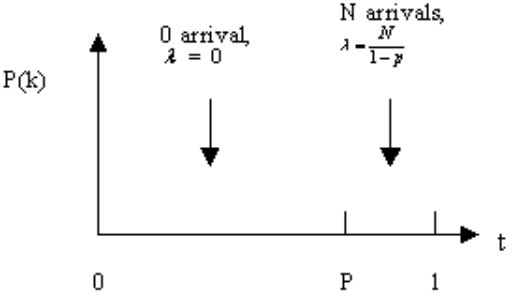


Fig. 2. Time segments assumption for periodical probing algorithm.

We can obtain the following results using queuing theory.
During time points p to 1:
The Poisson arrival velocity λ,

$$\lambda = \frac{N}{\Delta T} = \frac{N}{1-p} \tag{4}$$

Combine Equations 3 and 4, we can obtain ρ, ratio of usage.

$$\rho = \frac{\lambda}{\mu} = \frac{N}{1-p} e^{\frac{\sigma^2}{2}} \tag{5}$$

Further,

$$T_{qp2} = \frac{\frac{1}{\mu}}{1-\rho} = \frac{e^{-\frac{\sigma^2}{2}}}{1-\frac{N}{1-p}e^{\frac{\sigma^2}{2}}} \tag{6}$$

$$T_{wp2} = \rho T_{q2} = \frac{\frac{N}{1-p}e^{\sigma^2}}{1-\frac{N}{1-p}e^{\frac{\sigma^2}{2}}} \tag{7}$$

During time points 0 to p: $\lambda = 0$, then $\rho = 0$, and further,

$$T_{qp1} = 0 \tag{8}$$

$$T_{wp1} = 0 \tag{9}$$

Then the weighted average for T_{qp} and T_{wp} are:

$$T_{qp} = pT_{qp1} + (1-p)T_{qp2} = \frac{(1-p)e^{\frac{\sigma^2}{2}}}{1-\frac{N}{1-p}e^{\frac{\sigma^2}{2}}} \tag{10}$$

$$T_{wp} = pT_{wp1} + (1-p)T_{wp2} = \frac{Ne^{\sigma^2}}{1-\frac{N}{1-p}e^{\frac{\sigma^2}{2}}} \tag{11}$$

4.2 System Performance of Requirement-Based Probing Algorithm

For the requirement-based probing algorithm, the Poisson arrival velocity is,

$$\lambda = \frac{N}{\Delta T} = \frac{N}{1} = N \tag{12}$$

The service velocity is the same one described by Equation 5, then

$$\rho = \frac{\lambda}{\mu} = Ne^{\sigma^2}2 \tag{13}$$

Further,

$$T_{qr} = \frac{\frac{1}{\mu}}{1-e} = \frac{e^{\frac{\sigma^2}{2}}}{1-Ne^{\frac{\sigma^2}{2}}} \tag{14}$$

$$T_{wr} = \rho T_{qr} = \frac{\rho}{\mu(1-\rho)} = \frac{Ne^{\sigma^2}}{1-Ne^{\frac{\sigma^2}{2}}} \tag{15}$$

Now, we can derive two conclusions.

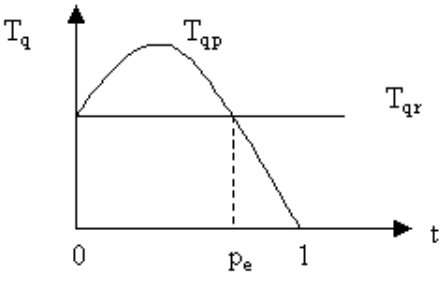


Fig. 3. Comparison of T_{qp} and T_{qr}.

Conclusion 1. $T_{qr} < T_{qp}$, $t \in (0, P_\epsilon)$.

Based on Equations 10 and 14, we can get the curves shown in Figure 3. where

$$P_e = \frac{1 - 2Ne^{\frac{\sigma^2}{2}}}{1 - Ne^{\frac{\sigma^2}{2}}} \tag{16}$$

Figure 3 shows that: if p locates in $(0, P_\epsilon)$ then, and if p locates in $(P_e, 1]$ then $T_{qr} < T_{qp}$. That means when the network load becomes heavy ($\sigma \uparrow$), or there are more customers ($N \uparrow$), or both of these events happen, then P_e becomes smaller. That is when the above situation(s) happen, in a system's view, T_{qp} is less than T_{qr}, but in practice, we hope that P_e is close to time point 1, that means we hope the resolver's database update period is only a small part of the whole time unit, because during $[P_e, 1]$, resolver will focus on database updating, therefore the performance of the service is poor. Based on the analysis, generally speaking, in most of the time unit, $(0, P_e)$, the performance of the requirement-based probing algorithm is better than that of periodical probing algorithm; only in a very small part of the time unit, $(Pe, 1)$, the former performance will be worse than the later.

Conclusion 2. $T_{wr} \leq T_{wp}$.

If $p = 0$ then $T_{wr} = T_{wp}$. This is easily obtained.

If $p > 0$ then $T_{wr} < T_{wp}$.

Proof:

$$\begin{aligned} p > 0 &\Rightarrow -p < 0 \\ &\Rightarrow 1 - p < 1 \\ &\Rightarrow 1 < \frac{1}{1-p} \\ &\Rightarrow -1 > -\frac{1}{1-p} \end{aligned}$$

$$\Rightarrow 1 - Ne^{\frac{\sigma^2}{2}} > 1 - \frac{N}{1-p}e^{\frac{\sigma^2}{2}}$$
$$\Rightarrow \frac{1}{1 - Ne^{\frac{\sigma^2}{2}}} < \frac{1}{1 - \frac{N}{1-p}e^{\frac{\sigma^2}{2}}}$$
$$\Rightarrow \frac{Ne^{\sigma^2}}{1 - Ne^{\frac{\sigma^2}{2}}} < \frac{Ne^{\sigma^2}}{1 - \frac{N}{1-p}e^{\frac{\sigma^2}{2}}}$$
$$\Rightarrow T_{wr} < T_{wp}$$

This shows that T_{wr} is always less than or equal to T_{wp}, namely the average waiting time of the requirement-based probing algorithm is always less than or equal to that of the period probing algorithm.

5 Remarks and Future Work

With the dramatic development of the Internet, anycasting will become an important part of forthcoming applications. Initially, anycasting was presented in the network layer, and application-layer anycasting is a practical choice for many current applications.

In this paper, we described the application-layer anycast model and the current algorithms for the critical part of anycasting, namely how to decide the "best" service server. We provided our requirement-based probing algorithm. The analysis shows that: 1) The average waiting time for all anycast queries of requirement-based probing algorithm is better than that of the periodical probing algorithm. 2) Generally speaking, in normal network situations, the average time used in system for an anycast query of the requirement-based probing algorithm is better than that of the periodical algorithm, except that the network load is very heavy, or there are too many anycast queries, or both of them happen.

Our next step is to perform simulations for both of the algorithms and try to obtain further confirmations about the advantages of the requirement-based algorithm. We will also try to explore the situations of multiple anycast resolvers and the synchronizations among the resolvers.

References

1. S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei, "Application-layer Anycasting," Technology report, College of Computing, Georgia Institute of Technology, 1996
2. S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei, "Application-layer Anycasting," IEEE INFOCOM'97, April, 1997
3. R. Caceres, "Measurements of wide-area Internet Traffic," Tech. Report. UCB/CSD 89/550, Computer science Department, University of California, Berkeley, 1989.

4. Jin Cao, William S. Cleveland, Dong Lin, and Don X. Sun, "On the Nonstationarity of Internet Traffic," Proc. ACM Sigmetrics '01, 102–112, 2001
5. S. Deering and R. Hinden, "Internet Protocol Version 6(Ipv6) Specification," RFC 2460, Dec. 1998.
6. J. Guyton and M. Schwartz, "Locating Nearby Copies of Replicated Internet Servers," in Proceeding of SIGCOMM 95, pp. 288–298, 1995.
7. R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 1884, Dec. 1995.
8. Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar, "A Novel Server Selection Technique for Improving the Response Time of a Replicated Service," IEEE INFOCOM'98.
9. Weijia Jia, G. Xu and W. Zhao, "Integrated Fault-tolerant Multicast and Anycasting Routing Algorithms", IEE Proceedings-Computers and Digital techniques, Vol. 147, No. 4, July 2000.
10. Weijia Jia, W. Zhou, and Joerg Kaiser, "Efficient Algorithms for Mobile Multicast using Anycasting Group", IEE Proceedings – Communications, Vol. 48, No. 1, February 2001
11. D. Katabi, and J. Wroclawsi, "A Framework for Scalable Global IP-Anycasting (GIA)," SIGCOMM'00, Stockholm, Sweden, Auguest, 2000
12. C. Partridge, T. Mendez, and W. Milliken, "Host Anycast Service," RFC 1546, Nov. 1993.
13. Vern Paxson, "Measurements and Analysis of End-to-End Internet Dybamics," Ph.D. thesis, University of California Berkeley, 1997
14. Vern Paxson, "End-to-End Internet Packet Dynamics," IEEE/ACM Transactions on Networking, Vol.7, No.3, pp. 277–292, June 1999.
15. Robbert V. Renesse, "Scalable and Secure Resource Location," the Proceedings of the Hawaii International Conference on System Sciences, Maui, Hawaii, January 2000.
16. Ricard V. Sole, and Sergi Valverde, "Information Transfer and Phase Transitions in a Model of Internet Traffic," Ricard V. Sole and Sergi Valverde, Physica A 289 595–605, 2001
17. Dong Xuan, Weijia Jia, Wei Zhao, and Hongwen Zhu, "A Routing Protocol for Anycasting Massages," IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 6, June 2000.

Experimenting with Gnutella Communities

Jean Vaucher[1], Gilbert Babin[2], Peter Kropf[1], and Thierry Jouve[1]

[1] University of Montreal
Computer Science and Operations Research, Montreal, Quebec, Canada
{vaucher, kropf, jouvethi}@iro.umontreal.ca
[2] HEC – Montreal
Information Technology, Montreal, Quebec, Canada
Gilbert.Babin@hec.ca

Abstract. Computer networks or distributed systems in general may be regarded as communities where the individual components, be they entire systems, application software or users, interact in a shared environment. Such communities dynamically evolve with components or nodes joining and leaving the system. Their own individual activities affect the community's behavior and vice versa. This paper discusses various practical experiments undertaken to investigate the behavior of a real system, the Gnutella network, which represents such a community. Gnutella is a distributed Peer-to-Peer data-sharing system without any central control. It turns out that most interactions between nodes do not last long and much of their activity is devoted to finding appropriate partners in the network. The experimental results presented have been obtained from a Java implementation of Gnutella running in the open Internet environment, and thus in unknown and quickly changing network structures heavily depending on chance.

1 Introduction

Whenever autonomous individuals act in a shared environment, the emergent interaction results in manifold relations between the individuals and/or groups of individuals. Those relations and the associated behavior of individuals and groups may induce structures to the groups. Such structures are commonly called *communities*. The behavior of biological individuals, such as ants or bees, but also humans has been widely studied in the social sciences [4]. Key findings include that despite the largely varying (intellectual) capacities of individuals and groups, a set of common characteristics for acting in a shared environment still may be observed [9]. However, this usually depends on the specific knowledge of the individuals and their time already spent within a community. Among the characteristics identified, we find that,

- each individual can identify a few members of a community and may exchange information with them;
- there is no single individual who knows or controls the whole community;

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 84–99, 2002.

- some individuals may be more "intelligent" than others and have more and/or better information;
- communities are often hierarchically structured with one or more outstanding individuals.

Starting with just a few individuals (at least two), communities continuously evolve. The resulting set of inter-related community members is generally called the *social network* of a community. While the number of individuals in a community can grow very fast, the single individual needs only little information about other individuals to still be able to potentially interact with a large number (or all) of the community members. The *six degrees of separation* property [14] illustrates this in the case of human communities. Moreover, communities are often characterized by a highly self-organizing behavior. Insect collectives such as ants or bees, but also physical and chemical systems composed of large numbers of individuals or particles interact locally and contribute thereby to global organization, optimization and adaptation to the environment.

Computer networks or distributed systems in general may be regarded as communities similar to the above examples. Most obviously, the Internet or Web forms entities that can be characterized as communities. Many approaches to define communities on the Web [6,8,10] are based on the use of existing link patterns and they therefore lack the characteristic community properties to adapt to the current context and to dynamically evolve. Implicit information [9, 13] other than link patterns are necessary to achieve this.

A number of applications have been developed which include in one way or another the idea of communities on the Internet. Among those are Yenta [7], an agent based system to find people with similar interests and to make them known to each other, Freenet [5], an information publication system storing, caching and distributing information on demand without any centralized control, or Gnutella [2], a distributed Peer-to-Peer data-sharing system. In the Gnutella system, a user only needs to know one (or several) other participants to join the community. The mechanism to broadcast information (search for partners or particular data) refrains from any central control: it is based on propagation of messages from participant to participant. The community is highly dynamic as participants can join and leave at any time without having to contact any administrative unit.

In order to investigate the behavior of communities on the Internet, the Gnutella system has been chosen for the research presented in this paper. The system provides an ideal practical testbed because all the participating individuals are unknown, no central control exists, and the community is sufficiently large. In fact, the only common component is the communication protocol and the core system behavior where each participant acts as a client and a server at the same time while applying the aforementioned information propagation mechanism.

The next two sections introduce the Gnutella system and protocol. In section 4 we describe the Jtella platform used for the experiments which are presented in detail in section 5. The method applied for measuring the performance

of Gnutella applications and the results observed are discussed in section 6 before concluding with a discussion about the various experimental results observed.

2 Overview of Gnutella

Gnutella is a distributed Peer-to-Peer (P2P) application for the sharing of files over the Internet. It was designed as a replacement for Napster [21] and has been used mainly for the dissemination of multimedia files.

Each participant in a Gnutella network runs a program on his computer that acts both as a client and a server as well as a router. Gnutella programs are referred to as *servents* (SERVer+cliENT), *nodes* or simply *clients*. As a client, the application provides an interface where a user can enter keywords describing the files that he is seeking. The program then sends the request to neighbouring participants who pass it on to their neighbours who do the same; thus propagating it throughout the network. At the same time, clients check to see if the request corresponds to local files they are willing to share and, if so, they send back a response. File transfers are done via another route using standard HTTP protocol requests.

The fundamental feature of Gnutella is that it does not rely on centralized databases or proprietary software. It also tries to ensure a measure of anonymity. As a result, it is resistant to both hardware failure and legal attack. The first Gnutella application was released in March 2000 but it was officially available for only a 24 hour period [2]. The basic protocol implicit in the original software is quite simple and is now available on the Web [1]. Although, it has been reported to suffer from performance and scalability problems [18], the Gnutella protocol has resulted in a large number of implementations.

Initially, as a replacement for Napster, the Gnutella network grew exponentially and this growth has been charted by several researchers [3,17]. Available data shows the network growing from around 1,000 nodes in November 2000 to over 40,000 in June 2001. Over this period, Ripeanu [17] found that over 400,000 different users had connected to Gnutella. In another study, a *crawler* programme found over 1 million different host addresses in an 8 day period [20]. However, since the summer of 2001, the network has been steadily shrinking, reaching an average of 16,000 users in January 2002 [3]. One can surmise that, if the main interest in Gnutella was sharing of music, many users have switched to more efficient specialised services such as Morpheus-KaZaA from MusicCity which now claims to have over 300,000 simultaneous users [15].

As the first widespread decentralised and open protocol, Gnutella is worthy of study. Its simple basic protocol also make it easy to use in experiments. Because the protocol is not specifically oriented to a single application domain (like mp3-encoded music), it is also easy to use Gnutella as a low level dissemination or broadcast protocol upon which to piggy-back other applications — with specially formatted query/response strings. Parallel private Gnutella networks can also be set-up by the simple expedient of using private bootstrap host caches.

3 Description of the Gnutella Protocol

Each participant in a Gnutella network maintains a small number of permanent links to neighbours (typically 4 or 5). Search is done via flooding — a distributed form of broadcast. Messages are sent to the neighbours who pass them on to their neighbours and so on. The number of hosts which are contacted in this way increases exponentially with each jump. In order to limit the potential data explosion, the number of jumps is bound by a time-to-live (TTL) counter which is decremented on each passing on. When the counter reaches zero, the message is no longer propagated. Messages also have a hop counter to keep track of how far they have come.

Gnutella provides also for some notion of anonymity. Specifically, queries do not contain the identity of the initiating host. Instead, each Gnutella message has a unique identifier (ID) and propagating hosts maintain routing tables keyed on this ID which indicate from which connection a message arrived. Answers carry the same ID and are returned along the same route as the query. The anonymity is only relative because downloads are done directly without passing through the Gnutella connections. The routing tables are also useful in preventing looping and duplicating messages: if the ID of a query (not an answer) is already present in the table, the message is seen to be a duplicate and is not propagated.

The Gnutella protocol is based on four types of messages (actually there is a 5^{th} type to deal with firewalls, but it is not pertinent to our discussion). The messages come in pairs: one for the requests and one for the answers. The file search pair includes:

Query – contains the user request as an unformatted string of keywords[1];
Reply – used by a host to return a list of matching files along with a short description of each file as well as the Host:Port address to be used for an HTTP download.

The next two messages are used to discover the addresses of participating hosts:

Ping – a request for host addresses;
Pong – a reply to the Pong with a Host:Port address along with extra information about the host bandwidth and the number of local files.

According to the protocol, a host receiving a Ping should answer with its own address in a Pong as well as forwarding the Ping to its neighbours. In practice, to reduce overhead, a host which already has too many neighbours, may pass on the message without returning its own address. Some hosts may act as central directories. They do not propagate Pings; rather, they maintain a cache of addresses they have received and return a small number of these. A number of sites well-known to the Gnutella community act as directories and this is how initial connection to Gnutella operates. However, any active host can serve as an initial connection point.

[1] Note: in possible extensions of Gnutella to specialized areas, one would expect the format and semantics of the Query request to be more tightly defined.

Finally, the protocol gives details about the handshake to be used on initial connection and suggests that Gnutella applications use port 6346 as the server address. The other aspects of the functionality that we expect in any program that accesses the Gnutella network either rely on other protocols or are left unspecified.

4 The Experimental Platform

Our experimental platform is based on Jtella, a framework written by Ken McCrary [11,12]. Jtella is made up of about 40 classes and 7000 lines of Java. It manages the initial connection to the Gnutella network, the maintenance of a specified number of connections and the routing of messages. An indication of the ease of use is that simple applications to search or monitor traffic require less than 150 lines of Java (on top of the Jtella Framework). Note that Jtella does not cover the indexing of files, matching queries or media playing. Jtella served as our introduction to the implementation of the Gnutella protocol but we took the liberty of rewriting or modifying about 1000 lines mainly dealing with parallelism and synchronization. We also uncovered, reported [22] and bypassed a Java bug: threads which are not started are not garbage collected.

4.1 Architecture of Jtella

The main building blocks of Jtella are: the Connection objects, the Router, the Connection managers and the Host Cache. The architecture is shown in Fig. 1. It is quite similar to that of LimeWire [19].

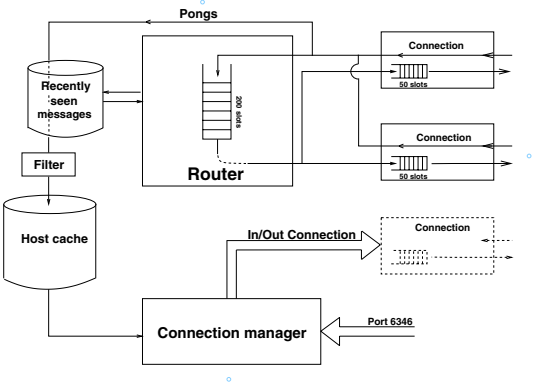


Fig. 1. Servent Architecture

There is one Connection object for each connection. Each Connection acts as a Thread to handle incoming messages and put them on a common message

queue for the Router. Each Connection also has a second Thread with a message queue to handle output messages.

The Router is a separate Thread. It takes messages off its queue, checks them against a table of recently seen messages and, if they are not duplicates, it places them in the appropriate output queues. The router tables are also used to return answers. Queues are fixed in length: the router queue has 200 slots and the output queues are 50 elements long. In general their occupation is less than 5%.

There are two connection managers whose job it is to maintain a specified number of active connections. Initially, Jtella was set up to keep 4 connections open: 2 outgoing and 2 incoming. If an incoming connection failed, the following incoming connection would be accepted. If the number of outgoing connections was below the specified level, for every missing connection, the outgoing manager would launch 2 start-up threads to try to connect to new hosts. The current version is more flexible in the split between incoming and outgoing connections. By default, two slots are reserved: one incoming and one outgoing, but the others can be of either type. When a connection fails, we launch two connector threads and, at the same time, we accept incoming connections. If all connection attempts succeed, we can temporarily have too many connections; but connections die quickly and this excess capacity is short lived. On start-up, all connections are necessarily outgoing but in a short time, as our address is made known through Pongs, the rate of incoming requests increases to the point that most failed connections are replaced by incoming connections.

To discover addresses of Gnutella participants, we send out a Ping whenever we open a new connection, and we put the addresses from all Pongs received into a cache. Because we receive many more host addresses than we can use, we limit our cache to 200 addresses and discard the others. As will be discussed later, most addresses that we receive are invalid. An important modification to Jtella was the addition of a filter to weed out bad addresses. When the cache is empty or low, we connect to well known host caches. These use the same Ping/Pong messages as all other Gnutella nodes but their sole function is to store addresses and redistribute them to later callers. Recently, this host cache function has been partially delegated to the network and in many servent implementation, whenever a node refuses a connection request, it sends back a number of Pongs from its host cache before shutting down the connection. Commonly, some servents send back 10 Pongs and others 50.

5 Gnutella Measurements

An important characteristic of Gnutella is that performance for any one session is highly dependent on chance. If a client happens to find reliable hosts early, it will obtain a steady flow of messages. At other times, it may struggle to find even a single permanent connection and it is not rare for identical servants run in parallel to have 2:1 differences in performance indicators.

Before proceeding to more exact measurements and tests, we present typical output from two exploration experiments which show the difficulty in maintaining connectivity and quantifying behavior.

5.1 Exploration Experiment I

Our principal measurement program named TestServent sets up a node with a specified number of connections, routes messages and collects statistics. It also prints out the current status of the node every 15 seconds. Typical output is shown in Figure 2.

```
****************  Fri Jan 04 10:01:33 EST 2002 **********************

Traffic in:  84 msg/s.
Valid   in:  22 msg/s.
Traffic out: 38 msg/s.

             Msgs:  Ping/Pong  -> Que/Rep
OUT ( OK )   1891:  335/998 -> 540/18  - cc652-a.plnfld1.com:6346
IN* ( OK )   220:  71/76 -> 73/0 - ACB51A11.ipt.aol.com:6349
IN  (temp)   1:  1/0 -> 0/0      - dmitry-pc4.la.asu.edu:47260
OUT (->? )   0:  0/0 -> 0/0      - 172.16.10.30:6355
IN  (temp)   0:  0/0 -> 0/0      - d15103.upc-d.chello.nl:2298
OUT (->? )   0:  0/0 -> 0/0      - 24.45.210.203:6346
OUT (->? )   0:  0/0 -> 0/0      - 62.70.32.25:6346
OUT (->? )   0:  0/0 -> 0/0      - 172.133.132.111:6346

Host Cache: 200  ==> Received: 2354, valid: 1116, used: 239

Threads:     59
   - SocketFactory Threads:     38
```

Fig. 2. Partial Output of Program TestServent

The first lines show the average traffic since the previous printout. The first thing to notice is that while 84 messages per second were received, most were invalid (either duplicates or Pongs with incorrect addresses); this left 22 valid messages which gave rise to 38 output messages.

Next, we see a snapshot of the connection activity. These are listed in the order in which they were created. OUT connections are created by our client, whereas IN connections were initiated by other hosts. For each connection, we give a status code (i.e., OK), list the number of messages received (total then categorized), and finally give the address of the corresponding host. In this test, we were trying to maintain 4 active connections, but at this moment there are 8 connections with only two in normal operation (OK). The other 6 connections are in various states of initialization or termination.

The first line shows the oldest connection, which has received 1891 messages and has been in operation for about 25 seconds. Only 18 messages are replies to queries. Typically, more messages are concerned with maintaining connectivity (i.e., Pings and Pongs) than with searching for information.

The second line shows an active *input* connection. The "*" indicates that the connected host has responded to our Ping with a Pong reporting its public

port number (6349). Of the other 6 connections, two, noted *temp*, are incoming connections which we decided to refuse (probably because 4 connections were up when they first arrived). We are keeping those temporarily open while returning some host addresses and waiting for them to answer our Ping. The other 4 are *output* connections in the process of opening a socket (`->?`).

The next line shows the state of the host cache. At present, it is full (200 addresses). 2354 Pongs were received but of those, less than half (1114) had valid distinct addresses. This number is more than enough because only 239 were used to open new outgoing connections or passed on to other nodes.

The last lines shows the parallelism (59 threads) required by Gnutella. It also underlines a problem with Java 1.3 whereby a thread trying to open a socket (our *SocketFactories*) may be blocked for up to 13 minutes before failing. In this case, the 38 Socketfactory threads include 34 blocked threads in addition to the 4 (`->?`) in the active list. The other 21 threads are involved in managing connections and routing messages.

This brief look at Gnutella underlines a fundamental aspect of the network: most connections do not last long and much of a client's activity is dedicated to finding replacements. In later sections, we will study this aspect more closely.

5.2 Exploration Experiment II

Figures 3 and 4 illustrate the stochastic nature of Gnutella. We ran two Gnutella sessions in parallel for 45 minutes and monitored two parameters: the number of messages received per second and the *horizon*, a measure of network size. More precisely, every minute, we broadcast a Ping and then we tally all the answering Pongs over the next 60 seconds.

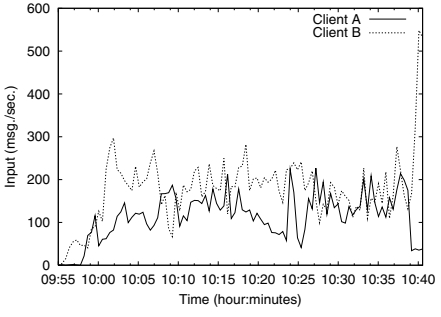


Fig. 3. Input Flow vs. Time for Two Clients

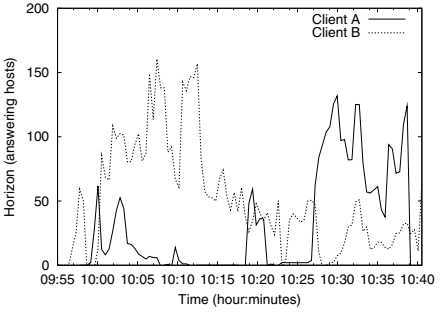


Fig. 4. Horizon vs. Time for Two Clients

Figure 3 shows the input rate while Figure 4 shows the horizon for the two clients. Both clients attempt to keep 4 connections open. The graphs are quite noisy but it is clear that Client B has done better than Client A. The average message rate for B is around 180 compared to 120 for A.

The random nature of operation is even more pronounced in the measurement of the *horizon*. It is hard to believe that these results were obtained from two *identical* programs run under *identical* conditions. This also shows the difficulty in trying to estimate the size of the Gnutella network.

In what follows, we present the results from experiments designed to quantify some critical aspects of Gnutella operation, namely:

- the validity of organizational information exchanged between nodes,
- the success rate in connecting to the network and
- the duration of sessions during which members actively participate in the network.

The experiments are presented in historical sequence as we tried to elucidate strange behaviour or make the implementation more efficient and robust. In all cases, we present results from 2 or more experiments to give an idea of typical behavior as well as variability. Although it is difficult to obtain exact meaningful measures of performance, our results still lead to interesting conclusions.

Later on, in Section 6, we present an experimental protocol that we used to overcome the stochastic nature of Gnutella in order to show the effect of an operating parameter, the number of active connections, on performance.

5.3 Validity of Pong Addresses

Open systems must deal with information received from many sources of untested quality. In the case of Gnutella, hosts depend on the Ping/Pong mechanism to discover the addresses of participating hosts. Unfortunately, early trials showed that many (if not most) addresses provided by Pongs are useless.

First, many addresses are duplicates. In experiments done around Oct. 16th, 2001, between 75 % and 88 % of addresses received were identical to addresses already present in our cache of 200 addresses. In one trial with 2390 Pongs received, a single address (32.101.202.89) was repeated 210 times. Fortunately, a simple test can be used to eliminate a large proportion of duplicates because about 25 % of addresses are identical to the one received immediately before.

Secondly, many addresses are "special" values (i.e. 0.0.0.0) which are obviously invalid. There are also blocks of Internet addresses which are reserved for particular uses and make no sense in the context of the Gnutella network. One example is multicast addresses but the most common problem results from hosts operating on private internets with NAT (Network Address Translation) translation [16]. These use addresses (i.e. 10.0.0.xx) which have no global validity. Table 1 shows the results from 2 experiments where we collected and analyzed all Pong addresses.

As a result of these experiments, we modified the cache algorithm in our client to filter out invalid and repeated addresses as well as those already in the cache. With these mechanisms in place, the data above shows that between 16 % and 28 % of addresses are retained. Due to the limited size of the cache, not all duplicates can be detected. The last line of Table 1 — the results of off-line

Table 1. Classification of IP Addresses Received in Pong Messages

	Experiment A		*Experiment B*	
Total addresses received	7482		19484	
Invalid addresses	2240	(30 %)	7042	(36 %)
Repeated addresses	1432	(19 %)	5298	(27 %)
Already in cache	1696	(23 %)	3978	(20 %)
Retained	2114	(28 %)	3166	(16 %)
Unique good addresses	1514	(20 %)	1792	(9 %)

analysis of all addresses received — shows the actual proportion of unique valid addresses to vary between 9 % and 20 %. Even with this drastic filtering and the use of a small cache, we normally receive many more addresses than we need.

```
Host Cache: 197  ==> Received: 59719, valid: 17489, used: 4145
```

5.4 Creating Sockets

Having filtered out invalid addresses, we then considered the probability of success in connecting to hosts whose addresses we retained. There are several reasons why a connection attempt could fail: the host may be too busy and refusing connections, the application may have terminated or the computer been disconnected from the network.

As mentioned previously, threads trying to open sockets to unavailable hosts remain blocked until the local system provides a timeout. In our set-up (Java 1.3 and Linux 2.2.17) this can take up to 790 seconds. In Windows environments, a smaller time-out of 45 seconds was reported. This delay has been a major source of inefficiency in both crawlers and servents; but the problem has been fixed in Java 1.4.

In one 90 minute session, our servent attempted to connect to 2541 hosts. Here is the breakdown of the results obtained and the average time to set-up the connection:

- 31 %: success - connection achieved in 2.3 sec.,
- 20 %: failure reported rapidly in 1.7 sec.,
- 49 %: blocked, failure noted after 10 sec.

To study this phenomenon more closely , we created a Connection tester (CTester) that takes a list of host:port addresses and tries to open a socket to each — which it then closes without attempting to do a Gnutella handshake. For each connection, it prints out the time until the socket creation terminates as well as the error message if the socket could not be created. For this test, we used 100 random addresses taken from a TestServent log file. Here are the results:

- 36 %: socket created in 1.6 sec. (9 sec. maximum),
- 26 %: rapid failure in 0.9 sec.,

– 38 %: blocking and failure reported after 790 sec.

The blocking during socket creation in Java explains the difficulty reported by several researchers who implemented crawlers to analyze the topology of Gnutella. Given the data above, where roughly, one connection attempt in three is blocked for 13 minutes, this means that a single thread can only examine about 4 addresses/minute and multi-threading is obviously a must.

5.5 Duration of Connections

We analyzed the log files from several sessions to determine how long connections stay valid once they have been established. In our longest test, maintaining around 7 connections over 24 hours on Nov. 28th, 2001, including 20,945 valid connections. By *valid*, we mean that a socket connection was established and the handshake was successful. At the same time, 36,000 incoming requests were refused and 6,000 outgoing socket creations failed. The average duration for all sessions was 31 sec. and the average set-up time was 0.21 sec. It is difficult to reason about an *average* connection, however, because the distribution is highly skewed and results are predicated by a small number of very large values. In this case, the longest session lasted about 11 hours (39,973 seconds) and 5 sessions lasted over 8 hours. Table 2 (Experiment C) gives an indication of the distribution of connection duration.

Table 2. Duration of Valid Connections

	Experiment C	*Experiment D*
Average	31 sec.	57 sec.
Median	0.17 sec.	0.4 sec.
Std. dev.	717 sec.	319 sec.
Max.	6350 sec.	3233 sec.
Average top 1 %:	2973 sec.	2960 sec.
Average top 10 %:	307 sec.	540 sec.
Average bottom 90 %:	0.26 sec.	2.3 sec.

In a more recent experiment, maintaining 5 connections over 1 hour on Dec. 30th, 2001, there were 297 valid Gnutella sessions for which the average set-up time was 1.05 sec. and the average duration was 57 seconds. Again the distribution was highly skewed and results are tabulated in Table 2 (Experiment D).

The main conclusion is that the average duration of a connection is quite short, between 30 seconds and a minute.

5.6 "Good" Hosts

Having determined that the majority of Gnutella participants are transients who only connect to the network occasionally and then for short periods, we then set

forth to see if the reliable hosts that we identified during one session could be reused in future sessions. If so, one could dispense with the need to connect to the same well-known host caches on start-up.

First, we extracted "good" connections from experiments done over 24 hours on December 30th, 2001. Our criterion for selection of a "good" host address was one to which the connection had remained active for at least 2 minutes (over twice the average connection duration). From 41,789 recorded connections, 564 connections (1.3 %) were considered "good."

Next, we scheduled periodic executions of the CTester program to see if it was still possible to re-establish socket connections to the "good" hosts. To obtain the public port for incoming connections, we send a Ping and waiting for a Pong with a hop count of 1. If they don't answer within a reasonable time, we assume the standard port 6346. Out of our 564 selected addresses, 191 (34 %) were incoming connections and of those only about a third (70) answered our Ping. In 75 % of these cases the public port returned was 6346; justifying our choice of that address for hosts that do not answer. Parenthetically, the fact that two thirds of our "good" hosts never responded to a Ping shows the difficulty in trying to measure network size by Pinging hosts!

The day after the addresses were obtained, we scheduled experimental runs every four hours over a 24 hour period. After this, we ran the experiment once a day for a week. During the first two days, the success rate for reconnection dropped steadily from about 18 % to 10 %. A week later, it reached 7 % where it has remained — varying between 6.4 % and 7.8 %.

This result may seem disappointing especially since in 380 cases (67 %) we were unable to reconnect even a single time. However, there were 4 hosts that we were always able to reach and another 57 who were available 50 % of the time or better. Additional experiments showed that we could open Gnutella sessions to 90 % of the hosts to which Ctester could open a socket. Thus it is possible to identify reliable semi-permanent Gnutella Hosts.

6 Measuring the Performance of Gnutella Applications

Beyond simply understanding the factors affecting the Gnutella network, our research is also aimed at improving the performance of applications. However, as demonstrated in our previous experiments, the performance of any one session depends on chance and measures of performance can therefore vary widely. Furthermore, the activity on the network varies with time. To be able to evaluate the effect of various servent parameters or strategies, we had to develop a methodology that would mitigate these problems.

The effect of random variation in performance can be reduced by running the servent over long periods, running multiple experiments at different times of the day and on different days of the week, and using averages from these runs.

However, it remains difficult to compare different algorithms. Clearly, we cannot compare two executions done at different time of the day or on different days, since there is no guaranty that the Gnutella network will be in the same

state. Our solution is to run test programs in parallel with a fixed benchmark and to consider the performance *relative* to the benchmark.

Another basic problem is choice of a measure of performance. Over the course of our study, we used several indicators:

- the total number of messages,
- the total number of Pings,
- the total number of Pongs,
- the average horizon, and
- the number of distinct host addresses found.

No measure stood out as a best indicator. As a result we used them all and gave them equal weight. This yields the following experimental methodology:

- for each parameter value (or strategy) that we wish to test, we run an experiment which lasts 24 hours and consists of 24 runs (of 45 minutes), once every hour,
- for each run, we launch two (2) servents in parallel, the test servent and a benchmark servent whose parameters are constant for all experimental runs,
- on each run, for each servent, we record the values of the 5 indicators listed above,
- the statistics collected serve to compute the *performance ratio*, noted r, of the test servent:

$$r = \frac{1}{m} \cdot \sum_{i=1}^{m} \frac{\sum_{j=1}^{24} x_{ij}^{t}}{\sum_{j=1}^{24} x_{ij}^{b}}$$

  where
  - m is the number of indicators used,
  - x_{ij}^{s} is the value of indicator i collected at the j^{th} run of servent s, where s equals b for the benchmark servent and t for the test servent.

We conducted a preliminary evaluation of this methodology to assess how the targeted number of connections (K) influenced the performance of a servent. We expected that performance should increase with K and perhaps taper off for very large values as bandwidth limitations start to play a role. We ran a series of experiments where the test servent used different values for the number of open connections. In particular, we had $K \in \{3, 4, 6, 8, 10, 15\}$ while the benchmark servant used a fixed value of five open connections ($K = 5$).

The results of the experiment are quite compelling and appear to be linear (Fig. 5)[2]. From this experiment, we observe that a servent with less than 2 target connections (1.68 to be exact) would not receive any traffic. It seems that 2 connections are always occupied trying to replace failed connections. Furthermore, there is no sign of tapering off; we could still increase performance by increasing K.

[2] $r = 0.276 \cdot (K - 1.68)$

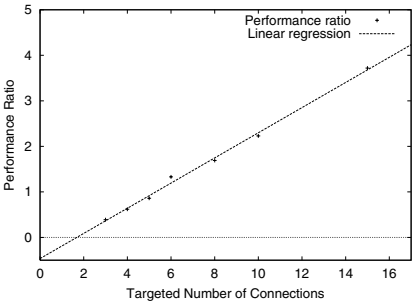


Fig. 5. Performance Ratio (r) vs. Targeted Number of Connections (K)

7 Conclusion

We have investigated the behaviour of participants in Gnutella, a well-known Internet community. Although some of the phenomena observed are particular to Gnutella, many of our results are relevant to other communities.

An important observation is the highly random nature of network behaviour. Repeatedly we observed a "whales and minnows" phenomenon whereby *average* measurements are determined by a small number of rare events with huge values and are therefore neither representative of the rare events nor of the more common small values. For example, we measured the average duration of a session to be 31 seconds, but 1 % of the sessions average 3000 sec. whereas the majority (99 %) average 1.3 sec. It is thus very difficult to get reproducible results.

Our experiments also showed that the composition of the community changes quite rapidly. Contrary to published results that suggest that connections last in the order of minutes or even hours [20], we found that sessions are much shorter: the duration of the average (median) session is less than half a second: 0.17 sec in one case and 0.4 sec in the other; and in another experiment we found that 98.7 % of the sessions lasted less than 2 seconds.

To maintain connectivity, nodes continuously exchange addresses of connected hosts that can be used to replace failed neighbours. A surprising observation was that a large proportion of the information thus obtained is incorrect or redundant: 80 to 90 % in the case of Gnutella's Pongs. A major part of the problem comes from hosts on sub-networks sending local (NAT) addresses which have no global validity. Even after filtering out flawed addresses, only a third of connection attempts result in a valid connection.

More generally, collaborative behaviour requires the exchange of organizational data between participants but flawed information may be a fact of life in open systems with unscreened participants, evolving technology, and a wide variety of software implementations.

Mapping the network or even estimating a *horizon* (the reachable portion of the net) may be more difficult than is generally believed. 2/3 of our "good" hosts never acknowledged a Ping; other nodes do not forward Pings to their

neighbours but return addresses from a local cache. The maximum horizon that we measured during our tests was less than 1000 nodes for 1 minute peaks. Average horizon values were much lower, normally in the hundreds.

We discovered semi-permanent reliable hosts but again they are rare. Starting from 42,000 site addresses we ended up with only 57 sites that were up 50 % of the time or better.

We developped an effective methodology — based on *comparative* measures and replication — to overcome the stochastic nature of network activity and allow the evaluation of various operating strategies.

In conclusion, the experimental investigation of Gnutella has revealed many interesting technical findings as well as conceptual insights. It became clear that a local intelligent screening and processing of community information is central for efficiency as well as scalability of such networks. Future work will thus concentrate on evaluating more sophisticated policies and strategies in both the real world of Gnutella and in simulated environments.

Acknowledgements. We would like to thank Cirano (Center for Interuniversity Research and Analysis of Organizations, Montreal) where J. Vaucher is on sabbatical leave, for providing the machines and environment for this research.

Matthias Klonowski, an exchange student from Rostock, Germany, worked on the Jtella software during the summer of 2001. He pointed out several performance bottle-necks and helped discover a critical Java bug [22].

References

1. The gnutella protocol specification v0.4. Clip2 Distributed Search Services. Available from `http://www9.limewire.com/developer/gnutella\_protocol_0.4.pdf`
2. What is gnutella. LimeWire LLC, 2001. Available from `http://www.limewire.com/index.jsp/what_gnut`
3. Gnutella network size (realtime graph). LimeWire LLC, 2002. Available from `http://www.limewire.com/`
4. E. Bonabeau, G. Theraulaz, J. Deneubourg, S. Aron, and S. Camazine. Selforganization in social insects. *Trends in Ecol. Evol. 188–193*, 1997.
5. I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, 2000.
6. Gary Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–160, Boston, MA, August 20–23 2000.
7. L. N. Foner. YENTA: A multi-agent referral system for matchmaking. In *First International Conference on Autonomous Agents*, Marina del Rey, California, 1997.
8. David Gibson, Jon M. Kleinberg, and Prabhakar Raghavan. Inferring web communities from link topology. In *UK Conference on Hypertext*, pages 225–234, 1998.
9. Francis Heylighen. Collective intelligence and its implementation on the web: Algorithms to develop a collective mental map. *Computational & Mathematical Organization Theory*, 5(3):253–280, 1999.

10. S. Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for emerging cyber-communities. *WWW8 / Computer Networks*, 31(11–16):1481–1493, 1999.
11. K. McCrary. The gnutella file-sharing network and java. JavaWorld, 2000. Available from `http://www.javaworld.com/javaworld/jw-10-2000/jw-1006-fileshare.html`
12. K. McCrary. Jtella: a java api for gnutella, 2000. Available from `http://www.kenmccrary.com/jtella/index.html`
13. V. Menko, D. J. Neu, and Q. Shi. AntWorld: a collaborative web search tool. In Kropf et al., editor, *Distributed Communities on the Web (DCW)*. Springer Verlag Berlin, 2000.
14. Stanley Milgram. The small-world problem. *Psychology Today*, 1967.
15. S. Osokine. Gnutella blown away? not exactly, 2001. Available from `http://www.openp2p.com/pub/a/p2p/2001/07/11/numbers.html`
16. Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address allocation for private internets. Request for Comments 1918, Internet Engineering Task Force, February 1996. Available from ftp://ftp.isi.edu/in-notes/rfc1918.txt.
17. M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. Computer Science Dept., University of Chicago, 2001. Available from `http://www.cs.uchicago.edu/files/tr\_authentic/TR-2001-26.pdf`
18. J. Ritter. Why gnutella can't scale. no, really, 2001. Available from `http://www.darkridge.com/~jpr5/doc/gnutella.html`
19. C. Rohrs. Limewire design. LimeWire LLC, 2001. Available from `http://www.limewire.org/project/www/design.html`
20. S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. Technical report UW-CSE-01-06-02, University of Washington, 2002.
21. J. Sullivan. Napster: Music is for sharing. Wired News, November 1999. Available from `http://www.wired.com/news/print/0,1294,32151,00.html`
22. J. Vaucher and M. Klonowski. Bug #4508232: Unborn threads are not garbage collected. Sun Developper Connection, 2001. Available from `http://developer.java.sun.com/developer/bugParade/bugs/4508232.html`

The SmartPhone: Interactive Group Audio with Complementary Symbolic Control

Tim Moors

School of Electrical Engineering and Telecommunications
University of New South Wales, Sydney, NSW 2052, Australia
t.moors@unsw.edu.au

Abstract. The SmartPhone provides a medium for distributed interactive group dialog by complementing an audio channel with a symbolic control channel. The control channel conveys information used for speaker identification, feedback, and turn taking. While these are conveyed visually in face-to-face meetings, their absence in purely audio systems limits the interactivity possible with such systems. Conveying control information symbolically avoids the bandwidth and other costs of video, while allowing novel modes of operation not possible in face-to-face meetings, e.g. anonymous feedback, prioritised turn taking and asynchronous skipping through meetings. The user interface to the control information is predominantly graphical.

1 Introduction

Projects increasingly involve the collaboration of multiple people, e.g. because they often require specialist knowledge from multiple disciplines. Consequently, individuals find that they are spending increasing amounts of time collaborating through such media as face-to-face meetings, audio- or video-conferences, and email. Unfortunately, these collaborations are often not as productive as we would like.

Productivity is partially determined by how the collaborations are conducted, but is also affected by the medium used, with each having certain disadvantages: Face-to-face meetings and videoconferences are interactive but require either physical relocation or high communications bandwidth, while email and audio-conferencing tend to be less interactive. The term "interactive" here means that the exchange is bidirectional, and that the turn-around time between a statement and corresponding response can be short. Textual exchanges stifle interactivity because typing retards the flow of information, they are usually message based (e.g. email, but not "chat" tools), and text does not support the richness of expression possible with video or audio. Audio-conferences are fine for disseminating certain information, but tend to decay into a clamour of voices when participants attempt to interact, as speakers start simultaneously and feedback interferes with speech. The transcript shown in Table 1 acts as an example of how inefficient an audio-only group collaboration can be.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 100–112, 2002.

Table 1. One third of an audio-only collaboration [1] conveys real content; the remainder is used for turn taking, speaker identification, or is wasted (typeface indicates category).

Time	Speaker	Speech
1	MMACS	*Flight, MMACS*
2	Flight	*Go,MMACS*
3	FDO	**MECO**
4	MMACS	**We're looking at what may be a**
5	MMACS and FDO (simultaneous)	small hydraulic leak Press-to-MECO
6	Flight	*Go ahead, MMACS. Say it again.*
7	MMACS	**a small hydraulic leak...**

A crude dichotomy of the information that flows in face-to-face meetings suggests why audioconferences fail to support interaction: The information can be classified as conveying either content or control. Content refers to the information that is explicitly conveyed, e.g. what is said and drawn, and is probably remembered after the meeting as the result of the meeting. Control refers to the information that regulates the exchange of content, e.g. identifying the speaker, deciding who can talk when, and sending feedback to the speaker. In face-to-face meetings, much of the content is conveyed audibly. When it is conveyed visually, e.g. drawings and shared documents, the object of focus for the vision is those repositories of information, not the participants. On the other hand, most of the control is conveyed by sight of the participant's faces and bodies. An audioconference may successfully convey the *content* of a meeting, but often falls short on conveying the *control* information.

The SmartPhone supports distributed group interaction by providing an audio channel and a complementary control channel. The control information is transmitted symbolically, rather than visually. This conserves bandwidth, compared to video, and can circumvent user reluctance to use video. By being an artificial construct, the symbolic embodiment of control information enables features not possible in face-to-face meetings. For example, feedback can be anonymous and turn taking can be prioritised. Participants access the control information through a user interface, which is predominantly graphical, as shown in Figure 1.

1.1 Structure of This Paper

The remainder of this paper is organized as follows: Section 2 examines the reasons for using audio, while the limitations of audio that limit the scope of the SmartPhone are covered in Section 3. Section 4 then describes the features of the SmartPhone in detail. Section 5 shows how the SmartPhone differs from related work. Section 6 concludes the paper.

2 The Future of Audio Is Sound

Audio currently enjoys advantages over other media for reasons that are both technological and sociological. While several technological advantages may diminish in the future, the sociological ones can be expected to remain. This section reviews these advantages of audio over competing media of face-to-face meetings, videoconferencing, and text.

In terms of technology, audio is at an advantage over competing media because face-to-face meetings take too long to set up (transportation technology), text is too slow to produce (might change with speech recognition technology), and videoconferencing imposes a heavy bandwidth load on current communications bearer technology. The bandwidth requirements of video streams can be reduced to some extent by compression, but this introduces delay, which can itself complicate turn taking or disrupt synchronisation with the audio [2,3]. Not only does each video stream require more bandwidth than an audio stream, but a meeting needs to convey more active video streams than audio streams. This is because only a few participants typically speak at any one time, whereas all participants continuously project their visual image. Video cameras also cost more than microphones.

Fig. 1. Core SmartPhone Graphical User Interface

Fig. 1. Core SmartPhone Graphical User Interface

The resemblance of videoconferences to face-to-face meetings often raises the expectations of users who later become disappointed when they experience (some times subconsciously) the shortcomings of videoconferencing. In particular, users expect to be able to use eye contact to address listeners, evaluate attentiveness, and determine who is looking at them, but despite the utility of gaze awareness [3], it can be difficult to establish in videoconferences involving more than two participants. The lack of gaze awareness can also cause participants to find videoconferences stressful because they don;t know when they are being watched, and so are forced to always behave as if they are being watched.

The fundamental advantage of audio over video or text is the mobility of transmitters. A microphone can be attached to the user, whereas a camera needs to be separated from a participant's face in order to capture it, and keyboards can be bulky and need a surface on which to rest. The richness of audio can make it a more engaging medium than text. And while audio is not as rich as video, it is often sufficiently rich to convey the necessary information and make the desired impact. Adding video can simply act to detract from the message,

especially from the sender's perspective. For example, when there is video, a participant may need to expend effort to maintain an image that they may have previously established (e.g. dressed in a suit), may find it harder to lie or deceive, and might be unable to conceal their other engagements. This degree of presence afforded by video is often considered to be invasive, and has hindered the acceptance of video systems such as AT&T's Picturephone.

While many of these issues in isolation pose only a small advantage for audio, in conjunction they are significant. This is because of the importance of network externalities [7] to communication services: the network becomes increasingly valuable to any one user as additional other users are attached. For example, the cost benefits of audio allow it to be deployed on desktops rather than dedicated conference rooms, providing greater accessibility and so greater penetration and greater value to users. Similarly, issues such as mobility and invasive presence may only directly affect a minority of users, but they establish a lowest-common denominator service that defines the user-base, and hence value, of audio systems such as the SmartPhone. The current prevalence of workplaces that are equipped with both a networked computer and audio communications (through a telephone, if not the computer) constitutes a vast user-base for a collaborative tool such as the SmartPhone. By interoperating with the Plain Old Telephone System, the SmartPhone can access an installed base of existing telephony customers.

3 Scope of the SmartPhone

The SmartPhone is intended to support tasks involving rational discussion between participants, such as might be encountered when generating ideas or plans, problem solving or decision making. Sight of participants' faces and bodies is important for some other tasks for which the SmartPhone may not be well suited. For example, a speaker may want to project their image to aid in persuasion, and may want to see the immediate response of listeners during a negotiation or interview. Video is also important for conveying emotions [5] and for creating a social presence [6]. Vision of the speaker's face can also help listeners interpret equivocal expressions, such as sarcasm or pauses in the audio [3]. Vision can aid comprehension, both because a listener can lip-read to reinforce what is heard [7] and because the speaker receives feedback about the extent of the listener's comprehension, which is important for trans-cultural exchanges [8]. The redundancy from multiple media can reinforce confidence in a judgement, albeit irrespective of the correctness of this judgement [9]. Vision may be useful at the start of a business encounter to establish social rapport, and at the end of the encounter to confirm agreement, but the SmartPhone targets the rational discussion in the body of an encounter.

The SmartPhone specializes in the medium of audio, with just enough complementary control information to make this medium suitable for effective group dialog. It is intended to be used in conjunction with tools that specialize in supporting other media, e.g. whiteboards, collaborative editing, and database

access tools. When a picture tells a thousand words, use a picture. The GUI is designed to be thin so that it can be located on the side of a screen that is shared with other tools. Some features of the SmartPhone might also benefit from closer integration with other tools. For example, textual communications may allow elaboration on anonymous feedback, feedback may tie in to voting tools, the turn taking may be coordinated with other floor control mechanisms, and the timing control should also affect other media needing synchronisation with the audio.

Group work is often classified according to its geographical or temporal distribution and the number of participants [10]. According to these criteria, the SmartPhone focuses on supporting distributed same-time collaborations between three to ten people. However, as a general-purpose audio tool, the SmartPhone supports a continuum from synchronous to asynchronous meetings (especially through the timing control described in Section 4.5), should also be useful for one-on-one meetings, and could extend to larger meetings where the timing features may be particularly useful. Furthermore, some of the benefits of the SmartPhone environment, e.g. anonymous feedback, may promote its use even when participants could have gathered together.

4 The SmartPhone

Software to demonstrate the SmartPhone concept has been implemented in C++ for the Microsoft Windows operating system, using the WAVE API for audio processing and Sockets for network communication. When a user starts running the SmartPhone on a terminal, they are asked to confirm their identity (inferred by looking up the address of the terminal in a table). Participants are subsequently identified by the address of the terminal that they are using, with an assumed one-to-one mapping between participants and terminals. The user can either initiate a new meeting, or join an existing meeting by responding to a personal invitation or public advertisement collected by monitoring a known multicast address. (In the future, the SmartPhone will use a standard session announcement protocol such as SIP.) The user who initiates a meeting is known as the Convenor of that meeting, and has special rights, such as the ability to remove other participants from the speaker queue. The Convenor's terminal is also responsible for advertising the meeting, distributing meeting state information (e.g. the state of the speaker queue) to participants joining the meeting, and for resolving disputes about the state of the meeting. Disputes might arise because each terminal maintains its own image of the state of the meeting(s) which its user is participating in, while control signals, which might cause state transitions, are multicast unreliably using UDP. Once the user is participating in a meeting, they interact with other participants through a meeting dialog box shown in Figure 1.

While communication tools can readily become filled with features to capture the nuances of communication, it is important that these features don't inundate a novice user, preventing them from using and learning about the tool. The

SmartPhone is designed to be familiar to existing telephone users, and introduce features in an incremental fashion. This is done by dividing the features into sets of aspects, each of which has its own "panel" of the user interface, which can be incrementally enabled, and allowing extra controls and displays to become visible to proficient users. In the description that follows, these aspects are described in an order that would form a common migration path for a user familiar with the telephone. In order, these aspects are the participants listing, participant identification, feedback, turn taking, and then timing. The treatment of each aspect starts with a description of how it is addressed in face-to-face meetings, how it is addressed in the SmartPhone, then some risks of the SmartPhone implementation.

4.1 Meetings and Participants Listing

In the face-to-face environment, the physical location of a person determines what meetings they can be participating in. While essentially everyone in the proximity can hear what is being said, multiple discussions can occur concurrently, with participants concentrating on select discussions by localising audio. Participants can visually determine who else is participating in the meeting.

In the SmartPhone, the meetings and participants panel, the second-lowest in Figure 1, lists the meetings that the user is participating in, and the participants in each meeting. The network infrastructure that the SmartPhone uses allows much richer connectivity than is available in face-to-face meetings. For example, private discussions can be made between arbitrary groups of participants, not just with immediate physical neighbours; participants can be prevented from speaking (e.g. novices listening to a discussion by experts) without resorting to social protocols; the communication can be asymmetrical, allowing listeners to audibly send feedback to the speaker without interfering with what other listeners hear. The SmartPhone allows a user to concurrently listen to multiple meetings, with the audio from these meetings mixed together. The user selects one of those meetings (shown highlighted) to specify which meeting the other panels of the SmartPhone refer to. Memory (a hierarchy of RAM and disk) appears as a form of meeting, allowing recorded dialogs to be played back and other meetings to be recorded. Memory is used to support the asynchrony features described in Section 4.5.

Participant names refer to either specific participants (e.g. "Joe Bloggs") or generic roles, which appear in parentheses (e.g. "(You)"). When a function, such as participant identification, is applied to a role, such as "(Speaker)", it is applied to various specific participants as they assume that role. The role of Speaker is governed by the turn taking system to be described in Section 4.4. Feedback symbols (described in Section 4.3) appear to the left of participant names, and feedback can be directed by clicking on the appropriate participant's name.

4.2 Participant (Speaker) Identification

In face-to-face meetings, participants may be introduced at the beginning of the meeting, and are subsequently identified by their physical appearance and location, and, for speakers, their vocal patterns and context (e.g. the FDO's "MECO" announcement in Table 1). Occasionally, speaker identification is irrelevant. Verbal introductions can bore participants who are familiar with the person being introduced, and can be forgotten by the time the person speaks.

The top panel of the SmartPhone user interface provides information identifying a participant chosen from the Participants listing. This contains the name and affiliation of the participant and an image of the participant. The image is particularly important since it is readily recognized, enhances the perception that the participant is a complete person, and provides something for the listener to focus their vision on. The description of the participant, as shown, is supplied by the participant themselves, but could equally include information that the user has compiled about the participant. The visual nature of the participant identification allows it be skimmed by users who are familiar with the participant, referred back to over time, and provides scope for greater speech compression since vocal patterns are not needed to identify a speaker.

4.3 Feedback

In face-to-face meetings, the audio channel tends to be dominated by one speaker, while other participants can visually send feedback through facial expressions, gestures, and posture. Audible feedback can be sent in one-on-one meetings, but in group meetings it tends to interfere with speech from the speaker. This feedback conveys the attentiveness of the listeners, their degree of comprehension, and other signals that disclose their response to what has been said.

Participants can send feedback by pressing on any of an array of buttons in the feedback panel. Feedback can be directed at a particular participant, e.g. the speaker, or sent to all participants in the meeting. When a participant receives directed and broadcast feedback from another participant, the directed feedback is displayed in the participants list on a blue background, masking the broadcast feedback. The two rightmost buttons of the feedback panel allow the user to send blank feedback (annulling any previous feedback that they may have sent), and to "stare" at another user, soliciting feedback from that user. A speaker can stare at select participants to gauge their attentiveness, and other participants may stare at each other to exchange responses. Staring must be directed, not broadcast, to prevent a starer from creating excessive amounts of work for other participants. Future work will add unobtrusive sensors to automatically augment this measure of attentiveness.

The presentation and meaning of the other buttons, and hence the feedback vocabulary, is arbitrary; they could be customized for each meeting. They are explained with textual "tips" that appear when the cursor is positioned over one of the buttons. The feedback buttons are arranged as a row of positive feedback above a row of negative feedback. From left to right, the columns of buttons

represent the listener's like (or dislike) of what is being said, their perception of its importance (garbage), request for elaboration (or compaction), and whether they understand or question it.

Feedback produced using this interface differs from feedback in face-to-face meetings in two important ways:

Volume: Listeners in face-to-face meetings can effortlessly produce continual feedback by way of their visual appearance, and it takes the speaker effort to look at them to obtain this feedback. In contrast, with the SmartPhone, the listeners must make an effort to produce the feedback. This calls into question what incentive listeners have for producing the feedback, perhaps reducing the *volume* of feedback.

Accuracy: Feedback in face-to-face meetings is often involuntary or subconscious: Some signals may be involuntary (e.g. pupil dilation), may be difficult to suppress (e.g. not yawning), and may be difficult to fake (e.g. an honest face when lying). In contrast, selecting a feedback icon is a deliberate act, which listeners can use to mask the feedback that is sent, or to send feedback that doesn't reflect their true state whether this be deliberate or accidental, e.g. by clicking on the wrong icon, or by the conscious measurement of their expression affecting their expression.

Little can be done about the accuracy of feedback, and this limits the utility of the SmartPhone for tasks such as negotiation and interviewing. The volume of feedback can be increased by making it easier to produce, e.g. through speech recognition of vocalisations, or ideally through automated gesture and expression recognition. The volume might also be increased by penalising participants who do not send feedback. One possible penalty is to inject into the audio that a participant hears some noise whose volume increases with the time since they last sent feedback. However, such a negative experience may be more detrimental than beneficial to SmartPhone meetings; often it is better to rely on social protocols for penalties.

An advantage of the computer-mediated SmartPhone over face-to-face meetings is that mediation can provide anonymity. This encourages participants to send feedback that they might have otherwise withheld, e.g. criticism or indications of their ignorance. Anonymous feedback is displayed in the participants listing under the "(Anonymous)" entry for the meeting, which, in turn, lists each type of active anonymous feedback with a tally of the number of participants who have sent that feedback. When a participant sends feedback anonymously, their identity is carried with the feedback to the SmartPhones of other participants, but is not displayed to those participants. This allows the SmartPhone to limit each participant to contribute at most one symbol to the anonymous feedback display, making it representative of the group sentiment. The SmartPhone also avoids disclosing consensus of participant feedback (which would destroy the anonymity) by limiting the displayed tally of receipts of an anonymous feedback symbol to two less than the number of participants in the meeting. The difference of two allows one participant to ask a question, and to see all but one of the other participants send the same anonymous feedback.

4.4 Turn Taking

In face-to-face meetings, turn taking is determined by a complicated social protocol [11], which is conveyed by such information as each participant's posture, facial expression, vocalisations, and the meaning of what has been said. More formal schemes, e.g. hand raising, are used in larger venues, where there are more participants and it is more difficult to see and hear subtle cues. When applied to networked terminals, regulated turn taking reduces the number of active audio channels, reducing the required bandwidth, the processing needed to mix channels, and noise levels.

The agility of turn taking is an important determinant of the interactivity of a medium. The original idea of turn taking with the SmartPhone was for participants who wish to speak to register their intent and form an ordered queue, with the SmartPhone blocking audio from everyone except the participant at the head of the queue. Such exclusive floor control is similar to that used in other collaborative environments, such as shared whiteboards [12]. However, it was soon realised that the mechanical movement (e.g. moving and clicking the mouse) required to change turns, in particular to relinquish a turn, was sluggish compared to the rapid changes which people are accustomed to in face-to-face meetings. To promote interactivity, the exclusive access was relaxed so that the participant at the head of the speaker queue (the "lead speaker") could admit other participants in the queue to an "interacting" level, whereby they could also be heard. Participants leave the interacting state either voluntarily, or by being demoted back to the queueing state by the lead speaker, or by being promoted to replace the lead speaker. Observations of face-to-face meetings indicate that only a few (e.g. two or three) participants need interact at any time, still providing much of the bandwidth, processing and noise benefits of exclusive access. Allowing multiple participants to share the floor complicates speaker identification; the participant currently in the role of Speaker can no longer be determined by the identity of the participant at the head of the speaker queue, but rather must be determined by comparing the power of the audio signals from interacting speakers.

In short, a participant usually progresses through the following turn taking states: Listening, Queueing, Interacting, and Leading. The controls available to a participant vary according to their state; Figure 1 shows the controls available to a Convenor when in the Leading state. The Convenor of the meeting has overriding control of the turn taking through a "dismiss speaker" button, which is unavailable to other participants, so that they can resolve problems such as a participant not relinquishing their turn. The meaning of the priority controls will be discussed shortly.

In initial trials of the SmartPhone, the audio was implemented using a PBX audio conference and so the turn taking system could not control its flow. Users tended not to adhere to the displayed speaker queue, but rather used it to create awareness of who wanted to speak, with social protocols determining who actually spoke. Rather than forcing users to take turns as directed by the SmartPhone, cooperative groups should be able to disable this control, with the

SmartPhone's speaker queue being suggestive rather than directive. Of course, when this is done, all potential speakers must emit audio continuously, requiring higher bandwidth and processing and introducing more noise than in the regulated case. Such suggestive turn taking may be particularly useful for meetings between participants with heterogeneous connectivity, where some participants meet face-to-face (where a computer can't control their behaviour) and others participate at a distance through the SmartPhone. Research into accommodating such meetings with heterogeneous connectivity is an important topic for further study.

In face-to-face meetings, participants can interject if they believe that what they have to say has sufficient priority. Such interjection accounts for the interjector's perception of the importance and urgency of what is to be said, and of the cost and opportunity cost of the interruption (e.g. the disruption and duration of the interruption compared to the time the person would otherwise have had to wait). Such perceptions are inherently subjective, with social protocols deterring a participant from repeatedly overvaluing their interjections. The SmartPhone provides two controls that allow a participant to modulate the importance and urgency of their turn taking request. Users should incorporate the duration of the disruption in their assessment of the urgency. The manual effort required to change these controls from their default state deters gratuitous use, and the name of a participant who has used these controls appears in the speaker queue with a prefix ("!!" for high priority, "_" for low priority) so that other participants can enforce retribution through social protocols if the prioritisation was unwarranted.

In addition to prioritisation according to *what* a participant wants to say, prioritisation can also be influenced by *who* wants to speak. Such speaker-based prioritisation may expedite the progress through the speaker queue of those with high rank, or retard the progress of those who have been excessively vociferous. The SmartPhone allows such prioritisation to be enabled on a per-meeting basis, with ranks determined from the table used to determine participant names from terminal addresses. Of course, any such mechanism is open to abuse, e.g. a highly ranked participant could domineer over lesser participants. The SmartPhone is not a social panacea, but provides options that people can use (or abuse) as they choose.

4.5 Timing

Note that the timing features described in this section are still being implemented.

In face-to-face meetings and traditional audioconferences, the medium has no memory. Yet, when a computer mediates communication, as with the SmartPhone, it can also store and process what is communicated. This has the potential to grant individual receivers great control, allowing them to time-shift information produced in the meeting so that it better matches their desires. In the SmartPhone, the timing controls allow the user to record, pause, and fast-forward dialog (audio and control).

Through recording, outgoing audio can be prepared, and incoming audio can be paused, reviewed and fast-forwarded. The ability to pre-prepare audio can reduce production blocking [10] by allowing participants to start talking as soon as an idea enters their mind rather than have to wait for their turn. A participant can *pause* what they hear of the meeting, providing them with time for such activities as writing notes, reflecting on the discussion, or attending meetings of higher priority.

Audio that has been recorded can be *reviewed*, perhaps repeating something for a listener who didn't correctly hear it the first time (important when there are many listeners), or allowing participants to speak freely without the burden of memorisation and later review what was said, e.g. to extract eloquent expressions [13]. Of course, a social protocol is also involved here: participants might become reticent if they fear that a recording of what they say could later be used against them.

A participant can also "*fast-forward*" over audio that has been recorded, allowing them to attend to only the items of interest. This has the potential to significantly improve participant productivity, since they can temporarily leave a meeting (e.g. for one of higher priority) and later rejoin the meeting, skimming over what they missed, pausing over what they find particularly pertinent, and eventually catching up to the live dialog between other participants in the meeting. The fast forwarding can be achieved by suppressing silent periods, playing the audio at a faster rate [14], or skipping over sections of audio. Of course, listener control comes at the expense of the speaker, who may be unable to use timing effectively for such purposes as emphasis.

Well labelled audio aids skipping. Labels can be generated automatically to indicate changes in speaker, but topical labels are also useful. These can be created by the user marking the dialog recording with annotations. These can be either private to this user, or publicized to all participants in the meeting. Another useful and simple way to create headings is to use pre-prepared text that will be discussed, such as an agenda for a meeting, or the slides of a presentation. When the discussion moves on to the next topic, the meeting convenor can mark the audio to synchronize it with the text.

5 Related Work

Telephone systems have been successfully used for dyadic audio communication for decades. However control signals are less important and problematic for dyads than for larger groups: in a dyad, there is only one speaker to identify and this is usually done at call setup, the full-duplex audio of the telephone allows the listener to send feedback ("uh huh") without interfering with what is being said, and there is limited competition for turns with the next turn always going to the solitary non-speaker.

Systems that carry telephony over packet-switched networks have been developed since the early 1980s (e.g. Etherphone [17]) and are currently the focus of great attention (e.g. see [18]). Such work is principally driven by the eco-

nomic advantages of packet telephony, and concentrates on the technical aspects of the audio, such as improving its fidelity, eliminating echoes and matching the volumes of multiple parties) [19]. In contrast, the aim of the SmartPhone is not to implement telephony using a computer, but rather to *augment* telephony by using a computer. Even multimedia groupware, such as the H.323 set of standards [20], only add audio to textual and graphical tools, rather than trying to enhance the audio by using other media. Some group packet audio tools (e.g. Thunderwire [21]) target serendipitous communications, whereas the SmartPhone targets deliberate meetings. The voice loops used by NASA [1] provide rich connectivity, but don't address the inability of audio to effectively convey control information.

Some commercial telephone-based audio-conferencing services provide supplementary control in the form of rudimentary question queueing and polling for feedback, as well as conference recording/playback. These are limited by the limited user interface afforded by the telephone handset. LearnLinc [22] provides some floor control and feedback, but, being targeted towards distance learning rather than meetings, provide less support for interaction than the SmartPhone.

6 Conclusion

The principal contribution of the SmartPhone is the recognition that effective group interactions require the conveyance of both content and control, and that audio alone is a poor conduit of control information needed for an interactive group meeting. With the SmartPhone, an audio channel is supplemented by a symbolic control channel. Apart from extending the interactiveness of audio channels towards that of face-to-face meetings, the control signals also enable features not possible in face-to-face meetings, such as anonymous feedback and prioritised turn taking.

References

1. J. C. Watts, et al.: "Voice loops as cooperative aids in space shuttle mission control", Proc. Computer Supported Cooperative Work, pp. 48–56, 1996
2. K. Ruhleder and B. Jordan: "Meaning-making across remote sites: How delays in transmission affect interaction", Proc. The 6th European Conference on Computer Supported Cooperative Work (ECSCW 99), Sep. 1999, pp. 411–30
3. J. C. Tang and E. A. Isaacs: "Why do users like video? Studies of multimedia-supported collaboration", Sun Microsystems, Technical Report TR-92-5, Dec. 1992
4. S. J. Liebowitz and S. E. Margolis: "Network externality" in The New Palgraves Dictionary of Economics and the Law, MacMillan, 1998
5. A. Mehrabian: Silent Messages: Implicit Communication of Emotions and Attitudes, 2nd ed, Wadsworth Pub. Co., 1981
6. J. Short, et al.: "Communication modes and task performance" in The Social Psychology of Telecommunications, pp. 77–89, 1976
7. V. Bruce: "The role of the face in communication: Implications for videophone design", Interacting with Computers, 8:166–76, 1996

8. E. S. Veinott, et al.: “Video matters! When communication ability is stressed, video helps”, Proc. ACM CHI 97 Conf. on Human Factors in Comp. Sys., pp. 315–6, 1997
9. G. D. Fowler and M. E. Wackerbarth: “Audio teleconferencing versus face-to-face conferencing: A synthesis of the literature”, W. J. of Speech Comm., 44:236–52, Summer 1980
10. J. F. Nunamaker, et al.: “Electronic meeting systems to support group work”, Comm. of the ACM, 34(7):40–61, Jul. 1991
11. H. Sacks, et al.: “A simplest systematics for the organization of turn-taking in conversation”, Language, 50:696-735, 1974
12. H. Dommel and J. J. Garcia-Luna-Aceves: “Floor control for multimedia conferencing and collaboration”, Multimedia Systems, 5:23–38, 1997
13. D. Hindus and C. Schmandt: “Ubiquitous audio: Capturing spontaneous collaboration”, Proc. CSCW 92, pp. 210–7, Nov. 1992
14. B. Arons: “Techniques, perception, and applications of time-compressed speech”, Proc. AVIOS ’92: C. of the American Voice Input/Output Society, pp. 169–77, Sep. 1992
15. A. Sellen: “Remote conversations: The effects of mediating talk with technology”, Human-Computer Interaction, 10(4):401–44, 1995
16. B. Freisleben, et al.: “The effect of computer conferences on joint decision making” in Human Aspects in Computing: Design and Use of Interactive Systems and Information Management, pp. 1123–7, H.-J. Bullinger (Ed.), Elsevier, 1991
17. D. C. Swinehart, et al.: “Adding voice to an office computer network”, Proc. Globecom, pp. 392–8, Nov. 1983
18. “Special issue on Internet telephony”, IEEE Net. Mag., 13(3), May/Jun. 1999
19. V. Hardman, et al.: “Successful multiparty audio communication over the Internet”, Comm. of the ACM, 41(5):74–80, May 1998
20. G. Thom: “H.323: The multimedia communications standard for Local Area Networks”, IEEE Comm. Mag., 34(12):52–6, Dec. 1996
21. D. Hindus, et al.: “Thunderwire: A field study of an audio-only media space”, Proc. ACM Conference on Computer-Supported Cooperative Work, pp. 238–47, 1996
22. LearnLinc Corporation: `http://www.learnlinc.com/`

PIÑAS: Supporting a Community of Co-authors on the Web

Alberto L. Morán[1,4], Dominique Decouchant[1], Jesus Favela[2], Ana María Martínez-Enríquez[3], Beatriz González Beltrán[1], and Sonia Mendoza[1]

[1] Laboratoire "Logiciels, Systèmes, Réseaux", Grenoble, France
{Alberto.Moran,Dominique.Decouchant,Beatriz.Gonzalez, Sonia.Mendoza}@imag.fr
[2] Ciencias de la Computación, CICESE, Ensenada, B.C., México favela@cicese.mx
[3] Depto de Ingeniería Eléctrica, CINVESTAV-IPN, D.F., México ammartin@mail.cinvestav.mx
[4] Facultad de Ciencias UABC, Ensenada, B.C., México

Abstract. To provide efficient support for collaborative writing to a community of authors is a complex and demanding task, members need to communicate, coordinate, and produce in a concerted fashion in order to obtain a final version of the documents that meets overall expectations. In this paper, we present the PIÑAS middleware, a platform that provides potential and actual collaboration spaces, as well as specific services customized to support collaborative writing on the Web. We start by introducing PIÑAS Collaborative Spaces and an extended version of Doc2U, the current tool that implements them, that integrate and structure a suite of specialized project and session services. Later, a set of services for the naming, identification, and shared management of authors, documents and resources in a replicated Web architecture is presented. Finally, a three-tier distributed architecture that organizes these services and a final discussion on how they support a community of authors on the Web is presented.

Keywords: Author Community, Web Collaborative Authoring, Doc2U, PICoS.

1 Introduction

A virtual community is an aggregation of individuals that share common interests or tasks and that use electronic media to communicate [10]. These communities have been classified based in the purpose they fit (e.g. social, informational, communicational, educational, for business, etc.), or the media they use (e.g. IRC, E-mail, or Web Communities). Also, they range from loosely-coupled communities that involve a great number of individuals that hardly know each other, and that mainly maintain short-term relations; to tightly-coupled communities where a relative small number of individuals, that know each other well, usually maintain long-term relations [11].

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 113–124, 2002.

From the simple need of "putting on paper" shared thoughts to the more specialized need of formally documenting the result of a join collaboration, the emergence of a community of coauthors is natural and certain. However, providing efficient support for collaborative writing to a community of authors is a complex and demanding task in itself, because members need to communicate, coordinate, and produce in a concerted fashion in order to obtain a final version of the documents that meets the expectations of them all [8][9][12].

The main goal of this work is the design and development of a collaborative environment to support a community of authors working together to produce common documentation in the widely distributed Web environment.

Previous research efforts have concentrated on the production of shared documentation by allowing the Web to be viewed as a writeable collaborative medium. WebDAV [13] extends the HTTP protocol by providing facilities for concurrency control, namespace operations, and property management of resources in a centralized server. Our previous work [2][3] proposes extensions to standard Web services providing mechanisms for the naming, identification, and shared management of authors, documents and resources in a replicated architecture. Both works provide mechanisms for robust document management on the Web.

Other research efforts have focused on providing support for shared virtual workspaces, and for the coordination of the authoring process, the allocation of responsibilities and the monitoring of progress. BSCW [1] provides shared virtual workspaces where users can upload documents, hold threaded discussions, and obtain information on the previous activities of other users to coordinate their own work. Contact [6] provides a framework, which uses Web forms, to let authors of a collaborative authoring project decompose the writing activity, and assign responsibilities for them. Doc2U [7] provides support to coordinate the collaborative writing activities by introducing the concept of document presence, and using it as a means to provide users with general awareness information of shared Web resources based on an instant messaging approach. All of them share the goal of providing general awareness of the co-authoring process.

In our approach, we aim to provide support for both, potential collaboration (finding someone to collaborate with, and making direct contact with him) and actual collaboration (building a common understanding, and executing, communicating and coordinating the actual work). To achieve this, we propose the use of PIÑAS Collaborative Spaces to integrate and structure a suite of specialized project and session services, along with services providing mechanisms for the naming, identification, and shared management of authors, documents and resources in a replicated Web architecture. In the remaining sections of the article, we introduce the PIÑAS Collaborative Spaces, present the other entities and services involved (Authors, Documents, Projects, Sessions, Applications and Events), and organized them in a three-tier distributed architecture, with a final discussion on how they support a community of authors on the Web.

2 The PIÑAS Collaborative Spaces: PICoS

Key concepts in our current collaborative environment, the PIÑAS Collaborative Spaces (or PICoS), can be thought of as virtual spaces to which users join to work collaboratively. A PICoS is shared among users, resources and applications, that are aware of each other, and that allows them to be easily contacted or located, as well as to communicate or exchange information among them.

PICoS are classified into Global, Initial and Focused PICoS. The Global PICoS (G-PICoS) is the whole collaborative space provided and managed by PIÑAS; is an aggregation of all Initial and Focused PICoS. Initial PICoS (I-PICoS) are used as a "rendez-vous" point where users meet with potential collaborators, creating or identifying potential collaboration opportunities with others sharing the Global PICoS. Focused PICoS (F-PICoS) are launched from I-PICOS by means of specific applications whenever the potential for collaboration is identified; they involve specific users, resources and applications, and are used to actually perform the collaborative work.

By having PICoS sub classified, we are able to separate initial and focused spaces from the global space, and to specify two specific interfaces to the PIÑAS API: the Initial and Focused API's. These API's can be implemented in PIÑAS-aware applications, and transparently used by non PIÑAS-aware applications, and can be implemented independently or together.

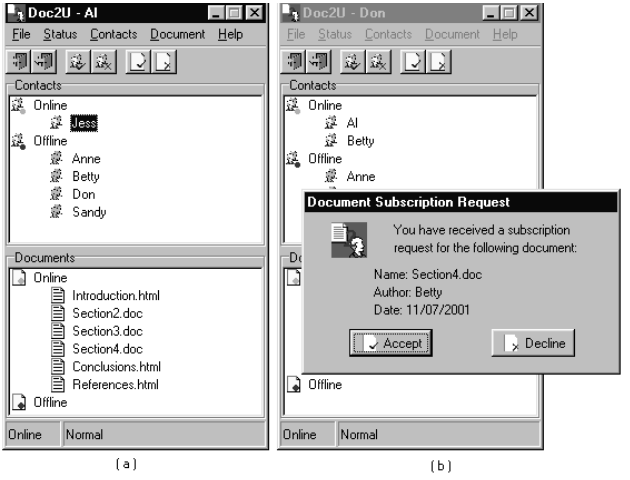


Fig. 1. Al's and Don's I-PICoS as presented by their Doc2U Clients

The current initial collaborative application used by PIÑAS, Doc2U, an Instant Messaging and Presence based tool, implements an I-PICoS by using the Initial API and allows coauthors working on shared resources to:

– Join a global PICoS that is shared among other users and resources,

- Be peripherally aware of other users and resources that share the global PICoS, based on a symmetric "publish and subscribe" mechanism,
- Send messages or commands to users and documents respectively,
- Launch other applications (with their corresponding focused PICoS) to interact with other users and resources.

To demonstrate the system in use, consider the following scenarios involving a small community of authors (*Al, Anne, Betty, Don, Jess, and Sandy*) distributed in several locations and working on the production of a joint paper. They are using Doc2U and the PIÑAS platform to support their collaborative work.

Even with the difference on location and time zones, they are able to get in touch and launch unplanned collaborative efforts without having to establish predetermined dates and times, because every time they get into their Doc2U client, the other subscribed users present in the global PICoS are notified of that in a similar way as other Instant Messaging and Presence systems, such as ICQ and AOL Messenger. This is shown in Fig. 1 (a), where user *Al* is notified of the arrival of user *Jess* to the current initial PICoS.

Whenever a new version of the documents is available, they submit it to the Document Repository, and after a successful submission, an automatic notification is sent to all coauthors: a *New Document Subscription* is sent if the document is new to the repository, or a *New Version Notification* is sent otherwise. Fig. 1 (b) shows *Don's* Doc2U client on receiving a *New Document Subscription* for document *Section4.doc* from user *Betty*.

Whenever someone obtains a document for editing, notifications of *Document Locked*, and with *Writing Status* are sent to all subscribed users. Fig. 2 (a) shows this in *Sandy's* client, when *Anne* is working on document *Section3.doc*.

Fig. 2 (b) shows *Jess'* Doc2U client when he launches a Chat application (and the F-PICoS) to have a conference with users *Al* and *Don*.

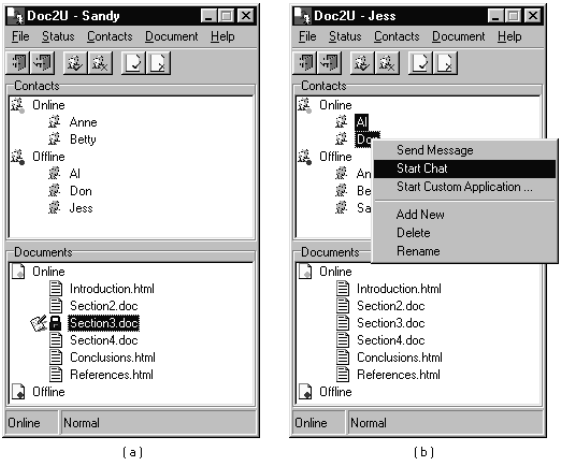


Fig. 2. Sandy's and Jess' I-PICoS as presented by their Doc2U Clients

3 The PIÑAS Platform

As previously presented, users are able to collaborate using initial PICoS, and then they can move their working scope to focused spaces PICoS. In both kinds of PICoS, basically they share entities on which they can act concurrently.

Thus, we have to analyze the different shared entities that are involved in these initial and focused collaborative spaces in order to highlight their inherent specific problems such as naming, distribution, replication, updating policy, integration of the contributions, notifications for group awareness facilities, supports for nomadicity and/or work in degraded mode. These issues will constrain the PIÑAS platform, and will allow us to adapt its architecture to more efficiently support collaborative authoring of Web documents.

3.1 Collaborators – Authors

The PIÑAS platform supports a specific naming principle that allows identifying users by giving them non re-allocatable Ids.

This way, PIÑAS can manage a distributed repository where author profiles are stored and that can be searched to "discover" new potential collaborators. Furthermore, it also allows the management of the working site where user-authoring actions are performed (see Fig. 3).

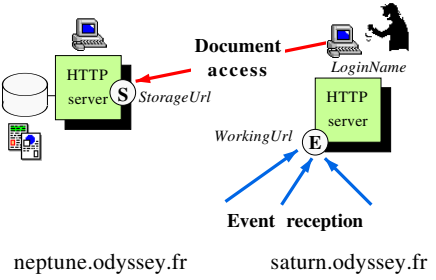


Fig. 3. Author Working and Storage Sites

As the proposed user naming system is based on standard Web technology, the identification of the authors follows the same principles. This approach constitutes the requested condition to extend existing Web services without creating a new, specialized, and heterogeneous naming system. Further detail is presented in [3].

3.2 Documents

Inside the PIÑAS storage layer, each document is represented by a set of files, which contain different pieces of information: fragments, author roles, state of each fragment, etc.

To allow each co-author to work on a shared document, even in case of a network failure, documents and their corresponding management are replicated on each repository, then, each author can work independently. Document consistency is based on a simple principle: in the whole system, there always exists one master copy for each fragment, which is the reference; there are as many slave copies as needed. On a given Web site where at least one author works on the document, all fragments of that document are available in local files and each fragment copy is either the master or slave copy for that fragment [3].

3.3 Projects and Sessions

Sessions and Projects are the representations of PICoS in the PIÑAS platform. They represent an ongoing instance of collaboration, and maintain the current status of each entity involved based on a collaborative relation. Based on the kind of PICoS that they represent, sessions are mapped to PICoS, thus, we have Initial and Focused Sessions that respectively map to initial and focused PICoS. The global PICoS represents the aggregation of all I-PICoS and F-PICoS and conceptually represents the aggregation of all initial and focused sessions, however, there is no unique entity to represent it as a global session in the system.

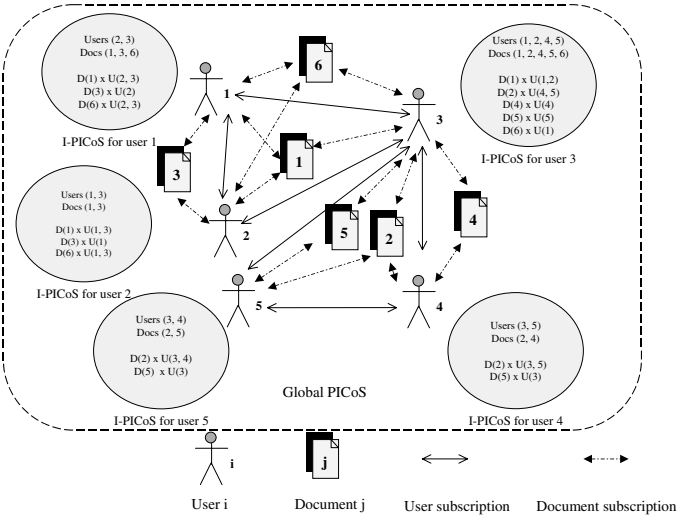


Fig. 4. G-PICoS and User Specific I-PICoS generated by the PIÑAS Platform

The information that is maintained in a session is the one that "dynamically" changes while users work on resources (documents) with specific applications, and the way this activity changes their state. This information includes [7]:

- *Availability status of participants, resources and applications,*
- *Activity status of participants, resources and applications,*
- *Content and/or meta-information modification of resources and applications,*
- *Applications.*

Projects represent the "long-term" definition of the collaborative effort in terms of the relations Users-Users and Users-Resources. They function as templates to build the global PICoS and initial sessions (corresponding to initial PICoS). Thus, the information that should be specified in a project includes [7]:

- *Users that will collaborate in the resource creation and modification process.*
- *Resources (Documents) that will be involved in the collaboration.*
- *User permissions and roles.*
- *Applications.*

In both cases (sessions and projects), references to applications are used to specify (even restrict) which applications can be used in the specific context, and to express which users can work with which applications, on which resources.

The Author, Document and Project definitions are used to generate the corresponding global PICoS and initial PICoS for each online user (see Fig. 4).

Whenever users detect the potential for collaboration and launch applications involving other specific users and resources, the system creates the corresponding focused PICoS, as shown in Fig. 5 for users 1, 2 and 3 working on document 1.

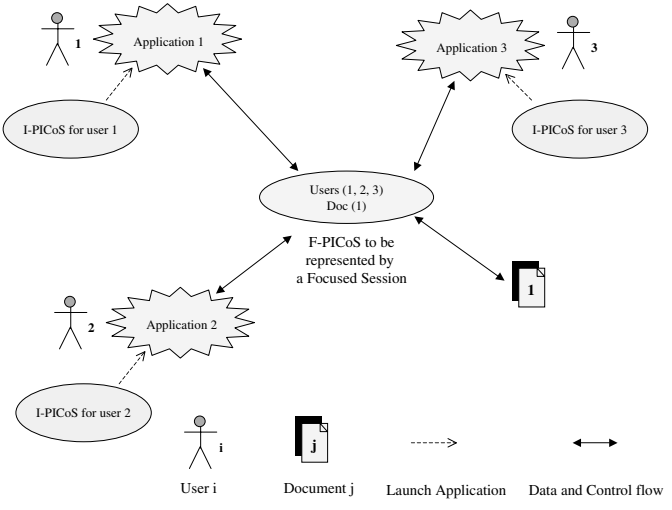


Fig. 5. Focused PICoS and their Launched Specific Applications

3.4 Applications and Events

Applications also constitute a kind of resource that needs to be identified. First, a dedicated application naming system has to be provided to uniquely identify and refer to the applications appearing as standard resources inside PICoS, Project and Session entities.

With the aim to define the system by which an application is uniquely identified by the coauthors on all cooperating sites, we have to take into account the following facts:

- The different sites may not work at the same time, so we cannot base the application naming system on a diffusion/agreement algorithm,
- For the same reason, a solution based on a unique and centralized application id allocator is not acceptable either.

The proposed solution consists in allocating a unique local identifier each time an author declares and installs a new application in the local PIÑAS environment (e.g. in a LAN, the environment composed of the local storage sites and the working sites where users may work from). An application can be locally installed and named without any agreement protocol with other remote PIÑAS environments. Of course, this independent application naming principle cannot ensure the unicity of the application Id in a more general Web collaborative environment. Thus, as shown in Fig. 6, the same application installed in several non-collaborating PIÑAS environments is identified with different Ids.

Later, if the authors of these systems want to establish a cooperation process, we propose to extend this basic naming system providing mechanisms that regularly diffuse the respective Application Naming Table (ANT) of each local environment to others. Each individual local environment will integrate the remote ANT in its own table defining a list of synonyms for each application. Each local ANT is managed as a standard cache of application synonyms that is regularly updated.

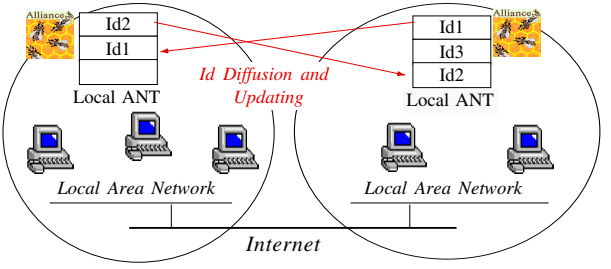


Fig. 6. Multiple Ids for the Same Application Entity

Events for Inter-Application Communication

Cooperative environments such as those that implement PICoS, typically include group awareness applications (or widgets) that are producers and/or

consumers of events. Events provide information or represent the user's work or actions: working focus (selection), type of action (cut, paste, copy, open, save), type of object (text, image, annotation), action dynamics (instant, frequency) and their specific attributes (color, bold, plan, importance). More particularly, collaborative authoring and reviewing tools, like AllianceWeb [2] and COARSY [5], are both producer of events and consumer of the events produced by other collaborative applications. Thus, the cooperative tool instance has to be aware of events generated by other collaborative applications to reflect and integrate their modifications in the author's working environment. Some applications may only be consumers, for example, a "radar view" whose goal is to show the author's focuses in a reduced representation of the document in quasi real-time mode in all working environments.

So, considering two users working on two different sites (see Fig. 7) who are respectively using the AllianceWeb cooperative editor and the "radar view" application: each authoring event has to be diffused both to the local user's "radar view" and to the remote author editor and "radar view".

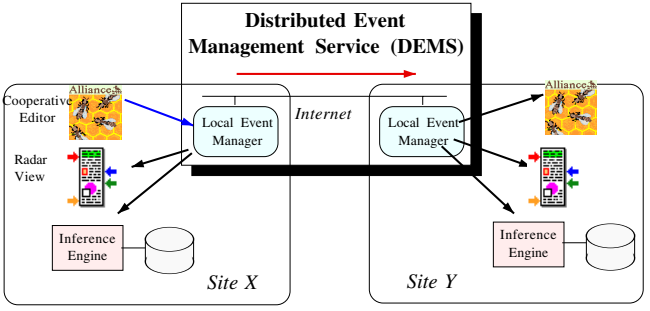


Fig. 7. Event Transmission between Distributed Cooperative Environments

Thus, a Distributed Event Management Service (DEMS) [4] is installed on each cooperating site: on each site a local server of this service acts as a concentrator of information (events) on the sender side and as a demultiplexer on the receiver side. For example, on the sender side (site X in Fig. 7), the "Local Event Manager" (LEM) receives events generated by the local producer applications, and if necessary it dispatches them to the local consumers. On the receiver side (site Y in Fig. 7), the LEM instance will determine the applications that are registered as event consumers and distributes the events to them.

This DEMS, developed in the scope of the PIÑAS platform is designed to support a Web cooperative authoring environment, and to do that we developed specific principles and mechanisms required by the development of various "group awareness" functionalities. These principles and mechanisms are dedicated to the management, the diffusion and the updating of the new application and event entities: a specific naming system allowing the unique identification of applications between the collaborating environments, and "production / trans-

mission / delivering" event system for establishing synchronous or asynchronous application communications.

3.5 The PIÑAS Architecture

Taking into account the above discussion on the PIÑAS entities and services, we propose a three-layer distributed architecture that integrates them (see Fig. 8).

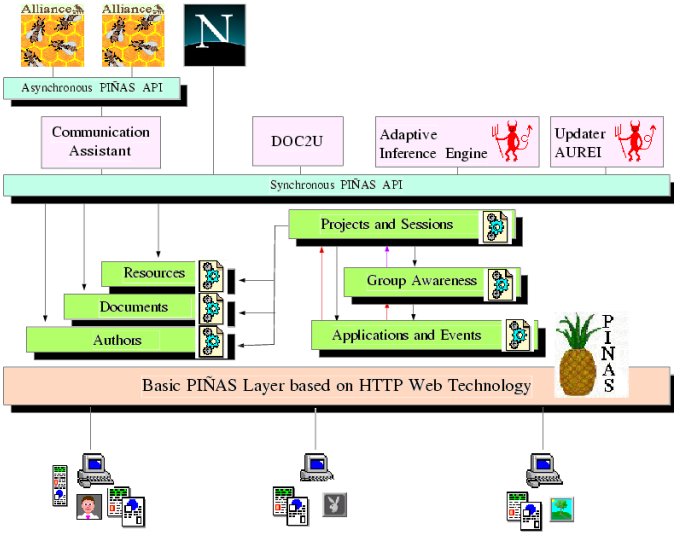


Fig. 8. The PIÑAS Architecture

Initial or focused collaborative applications (upper layer) are built on top of the PIÑAS platform and use its services. In the bottom layer, we found a set of basic collaborative entities that are managed by the (extensible) PIÑAS services (middle layer). Further decomposition of each layer, as depicted in Fig. 8, shows individual modules for the management of the previously introduced entities: Projects and Sessions, Authors, Documents and their associated Resources, as well as Application and Events. An additional service module is added to take in charge the management of the "Group Awareness" functionality.

4 Discussion and Conclusion

In this work, we aim to provide potential and actual collaborative authoring support for a community of authors on the Web.

Regarding potential collaboration we have introduced:

- Initial PICoS (I-PICoS), used as "rendez-vous" points, where authors can meet with potential collaborators, creating or identifying potential collaboration opportunities with others sharing the Global PICoS (G-PICoS). Also, they provide with shared resource presence information (availability, status, activity, etc.) from where additional potential collaboration opportunities can be identified and initiated.
- Author identification, naming and management allows for uniquely identified authors throughout the G-PICoS as well as to provide a "discovery" mechanism to search for someone to collaborate with.
- Document and resource naming and management allows for uniquely identified documents and resources in a distributed storage space throughout the G-PICoS, to provide an access mechanism to indirectly find someone to collaborate with, and to provide presence information on shared resources.
- Project and Initial Sessions, that map into G-PICoS and I-PICoS, to initialize the Potential collaborative space.
- Initial Applications that implement (completely or partially) the initial API and that provide the initial functionality ("rendez-vous" point and potential collaborative space).

Additionally, concerning actual collaboration we have introduced:

- Focused PICoS (F-PICoS) and Focused Sessions where the collaborative work is actually performed, that involve only the specific users, resources and applications, and that provide focused awareness on the specific task.
- Focused Applications, those that implement (completely or partially) the focused API and that provide specific or task oriented functionality in one or more of the required collaborative axes: communication, coordination and production.
- A dedicated Distributed Event Management Service (DEMS) that allows for Event transmission from a producer application to the (potentially distributed) consumer ones, and that forms the base of the peripheral and focused awareness service.

Finally, we have integrated all these entities as services in a three-tier distributed architecture that allows a community of authors (working from different sites) contact each other by means of an initial application implementing an initial PICoS. From there, the community obtains all the information required to identify potential collaborators, and seamlessly move from a private to a public space by launching specific applications (with their corresponding focused PICoS) to collaborate with other users sharing the global collaborative space.

Acknowledgments. This work is supported by: ECOS / ANUIES project M98M01, CONACyT projects 29729-A and 33067-A, CNRS / CONACyT projects 9018 / E130-505 and 10395 / E130-706, and SEP-SESIC / UABC and CONACyT with the scholarships UABC-125, 110630 and 118380 respectively provided to the first, fifth and sixth authors.

References

1. W. Appelt, "WWW Based Collaboration with the BSCW System", pp. 66–78, *in Proc. of SOFSEM'99,* Lecture Notes in Computer Science #1725, Springer Verlag, Milovy, Czech Republic, November 26–December 4 1999.
2. D. Decouchant, A. M. Martínez and E. Martínez, "Documents for Web Cooperative Authoring", *In Proc. CRIWG'99, 5th CYTED-RITOS International Workshop on Groupware,* pp. 286–295, IEEE Computer Society, Cancun (México), 15–18 September 1999.
3. D. Decouchant, J. Favela and A. M. Martínez-Enríquez, "PIÑAS: A Middleware for Web Distributed Cooperative Authoring", *In Proc. SAINT'2001, The 2001 Symposium on Applications and the Internet,* pp. 187–194, IEEE Computer Society and IPJ Information Processing Society of Japan, San Diego, California (USA), 8–12 January 2001.
4. D. Decouchant, A. M. Martínez, J. Favela, A. L. Morán, S. Mendoza and S. Jafar, "A Distributed Event Service for Adaptive Group Awareness", *In Proc. of MICAI'2002,* pp. 506-515, Lecture Notes in Artificial Intelligence #2313, Springer Verlag, Mérida, México, 22-26 April 2002.
5. J. Favela and D. Ruíz, "Collaborative Authoring and Reviewing over the Internet", *In WebNet Journal: Internet Technologies, Applications & Issues,* vol. 3, num. 3, pp. 26–34, 2001.
6. A. Kirby, and T. Rodden, "Contact: Support for Distributed Cooperative Writing", *in Proc. of ECSCW'95,* pp. 101–116, Stockholm, Sweden, September 10–14 1995.
7. A. L. Morán, J. Favela, A. M. Martínez and D. Decouchant, "Document Presence Notification Services for Collaborative Writing", *In Proc. CRIWG'2001, 7th International Workshop. on Groupware,* pp. 125–133, IEEE Computer Society, Darmstadt (Germany), 6–8 September 2001.
8. C. M. Neuwirth, D. S. Kaufer, R. Chandhok and J. H. Morris, "Computer Support for Distributed Collaborative Writing: Defining Parameters of Interaction", *In Proc. of CSCW'94,* pp. 145–152, ACM Press, Chapel Hill, NC (USA), October 1994.
9. I. R. Posner, and R. M. Baecker, "How people Write Together", *In R.M. Baecker (Ed.), "Readings in Groupware and Computer-Supported Cooperative Work: Assisting Human-Human Collaboration",* pp. 239–250, Morgan Kaufman, San Mateo, CA (USA), 1993.
10. H. Rheingold, *"The Virtual Community: Homesteading on the Electronic Frontier",* Addison-Wesley Publishing Company, Reading, MA, (USA), 1993.
11. J. Schilchter, M. Koch, and C. Xu, "Awareness - The Common Link Between Groupware and Community Support Systems", *in T. Ishida (Ed.), "Community Computing and Support Systems",* pp. 77–93, Lecture Notes in Computer Science #1519, Springer Verlag, Berlin Heidelberg, 1998.
12. M. Sharples, J. S. Goodlet, E. E. Beck, C. C. Wood, S. M. Easterbrook and L. Plowman, "Research Issues in the Study of Computer Supported Collaborative Writing", *In M. Sharples (Ed.), "Computer Supported Collaborative Writing",* pp. 9-28, Springer Verlag, 1993.
13. E. J. Whitehead Jr., and Y. Goland, "WEBDAV: A Network Protocol for Remote Collaborative Authoring on the Web", *In Proc. of ECSCW'99,* pp. 291-310, Copenhagen (Denmark), 12–16 September 1999.

A Group-Based Time-Stamping Scheme for the Preservation of Group Intentions

Liyin Xue[1], Mehmet Orgun[1], and Kang Zhang[2]

[1] Dept of Computing, Macquarie University
Sydney, NSW 2109 Australia
{lyxue, mehmet}@ics.mq.edu.au
[2] Dept of Computer Science, University of Texas at Dallas
Richardson, TX 75083 USA
kzhang@utdallas.edu

Abstract. Real-time negotiation support is an important issue in collaborative editing systems. The existing multi-versioning techniques provide support for negotiation by preserving individual users' concurrent conflicting intentions in multiple versions. They are "intrusive" in the sense that the created versions are embedded into the document under editing. This paper proposes a real-time annotative negotiation approach, where the document is annotated rather than embedded with various versions of the objects in conflict. It focuses on the way in which a group consensus or intention can be integrated into the document. The major technical issue is the time-stamping of the operations contained in the group intentions. A group-based time-stamping scheme is proposed to time-stamp them such that individual intentions and group intentions can be treated in a consistent manner.

1 Introduction

A real-time distributed group editor is a system that allows a distributed community of users to simultaneously edit a shared document [2,8]. In a wide area network environment, like the Internet, a replicated architecture is usually adopted such that the responsiveness of users' actions can be guaranteed, where all information is replicated on each user's workstation that runs a copy of the application process. Each process communicates directly with processes running on other users' computers.

With real-time group editors based on the Internet, the WYSIWIS (What You See Is What I See) property can not be guaranteed, that is, each user can not instantly see changes made by other users to shared objects, so users may happen to edit the same part of the document at the same time if no restrictions on editing are introduced [6]. Therefore, the major challenge of supporting responsive and unconstrained cooperative editing is the management of multiple streams of concurrent activities so that document consistency can be maintained in the event of conflicts.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 125–137, 2002.

The intention of an operation is the execution effect that can be achieved by applying it to the document state from which it was generated. The intentions of a user are represented by the intentions of the operations that the user issues. In group editors, a direct consequence of unconstrained editing is intention violation due to concurrent executions of operations, e.g., one intention may be overwritten by another. Therefore, intention preservation is an important aspect of consistency maintenance in group editing systems [8,10,11,12,13,14].

The existing consistency models are predefined for systems implementing them and the consistency can be guaranteed by the systems. However, only via non-predetermined negotiation about the conflicts among participating users, may synergy happen. It is generally infeasible for the system or individual users to know the emergent synergetic negotiation results or group intentions in advance. Therefore, the systems must be capable of reflecting all aspects of human conflict and cooperation and facilitating human users in conflict resolution. To put it in another way, the systems must preserve individual users' intentions that may be conflicting with each other, and provide facilities to help users reach a consensus or group intention [12]. Multi-version approach has been introduced to accommodate individual users' concurrent conflicting intentions in multiple versions in Tivoli [5] and GRACE [10] systems. Post-locking approach has been proposed to control the proliferation of versions and facilitate the process of negotiation [12]. This paper presents a group-based time-stamping scheme for the preservation of group intentions.

The rest of the paper is organised as follows. We start with an examination of the existing multi-versioning techniques in supporting negotiation and propose our new approach in Section 2. In Section 3, a representative-based time-stamping scheme is proposed to time-stamp those agreed-upon (or *team*) operations such that group intentions can be preserved. Section 4 introduces new temporal relationships necessary for team operations in correspondence with those for individual users' operations. Section 5 then proposes a new group-based time-stamping scheme. Finally, Section 6 summarises the major contributions of the paper.

2 Related Work and Our Approach

In a distributed system, it is sometimes impossible to say that an event occurred before another event, since the physical clocks are not perfectly accurate and do not keep precise physical time [3]. Causal precedence relationships among events happened at different processes can only be determined relatively. In a collaborative system, such relativity goes even further. Causal precedence relationships are based on human users' perception on the order of events instead of a perfect physical clock. Since both multi-version approach and our group intention preservation scheme are based on such a relative time, this section first introduces several temporal relationships between operations generated by individual users, then identifies the weaknesses of the existing multi-versioning techniques and proposes our new approach.

2.1 Temporal Relations of Individual Operations

We presume a given finite set OP of operations of a collaborative editing session and a set of collaborating sites: $Site = 1, \ldots, N$, where N is the number of sites.

In accordance with Lamport's solution [3] to the problem of physical clocks in distributed systems, a causal ordering relation of operations can be defined as follows [11].

Definition 1. Causal ordering relation $\rightarrow$*: Given any operations O_a and O_b in OP, generated at sites i and j respectively, O_a is said to be* causally preceding O_b*, denoted by $O_a \rightarrow O_b$, if (1) $i = j$, and the generation of O_a happened before the generation of O_b, or (2) $i \neq j$, and the execution of O_a at site j happened before the generation of O_b, or (3) there exists an operation O_x, such that $O_a \rightarrow O_x$ and $O_x \rightarrow O_b$. Where the relationship* happen before *is well defined in terms of local physical clocks.*

Definition 2. Dependent and independent operations*: Given any operations O_a and O_b in OP, (1) O_b is said to be dependent on O_a if $O_a \rightarrow O_b$. (2) O_a and O_b are said to be independent (or concurrent), expressed as $O_a || O_b$, if neither $O_a \rightarrow O_b$ nor $O_b \rightarrow O_a$.*

Obviously, $\rightarrow$ is a partial order relation.

To capture the causal relationships among operations in distributed collaborative editing systems, a time-stamping scheme based on a vector clock, i.e., state vector, has been proposed [2,11]. We redefine it as follows so that it is consistent with our new scheme to be presented.

Definition 3. Time-stamping scheme*: Let N be the number of cooperating sites in the system. Assume that they are identified by integers $1, \ldots, N$. Each site maintains a state vector SV with N components. Initially, $SV[i] := 0$, for all $i \in \{1, \ldots, N\}$. After executing an operation generated at site i, the system updates the clock as follows: $SV[i] := SV[i] + 1$. Whenever an operation O is generated at site k, it is first time-stamped with a state vector SV_O and then executed at the local site immediately and multicast to remote sites, where $SV_O[k] = SV[k] + 1$; $SV_O[i] = SV[i]$ ($i \in \{1, \ldots, N\}$ and $i \neq k$); where SV is the state vector of site k.*

By zooming in on the whole process of operation generation and execution, we can see the following set of consecutive actions performed by the system: operation generation, operation time-stamping, operation execution at the local site, local clock updating, operation propagation to remote sites, operation execution at remote sites, clock updating at remote sites. The temporal relationships of operations can be determined by comparing their state vectors [9].

For any pair of operations O_a and O_b, if $O_a \rightarrow O_b$, then O_a is generally required to be executed before O_b at all sites. This is called the *causality-preservation property* [11]. The operations received by a site may be out of order. An operation received from any remote site is causally ready to execute only when all the operations preceding it have been executed at the same site. This can be formally defined as follows [11]:

Definition 4. Causally ready*: Let O be an operation generated at site s and time-stamped by SV_O. O is causally-ready for execution at site d ($s \neq d$) with a state vector SV_d only if the following conditions are satisfied: (1) $SV_O[s] = SV_d[s] + 1$, and (2) $SV_O[i] \leq SV_d[i]$, for all $i \in \{1, \ldots, N\}$ and $i \neq s$.*

The execution order of independent operations may be different at different sites. Sometimes it may be desirable to have a global consistent execution order. A total ordering relation has been defined for this purpose as follows [11]:

Definition 5. Total ordering relation $\Rightarrow$*: Given any two operations O_a and O_b in OP generated at sites i and j and time-stamped by SV_a and SV_b, respectively, $O_a \Rightarrow O_b$, if (1) $\text{sum}(SV_a) < \text{sum}(SV_b)$; or (2) $i < j$ when $\text{sum}(SV_a) = \text{sum}(SV_b)$, where $\text{sum}(SV) = \sum_{i=1}^{N} SV[i]$ and N is the number of participating sites.*

Apparently, the total ordering relation is consistent with the causal ordering relation in the sense that if $O_a \rightarrow O_b$, then $O_a \Rightarrow O_b$.

2.2 Embedded Multi-versioning

In object-based cooperative editing systems, when two concurrent operations change the same attribute of the same object to different values, intention violation occurs. It is impossible for the system to accommodate the conflicting intentions in the same targeted object. The only way to preserve the intentions of both operations is to make new versions of the object, and then apply the two conflicting operations to the two versions separately. The execution effect is that no conflicting operations are applied to the same version. This is the multi-version approach in Tivoli and GRACE.

In a highly concurrent distributed environment, there may exist lots of operations involved in the conflicts and multiple versions will be created. Each version consists of a set of compatible operations. The versions created are mixed up with the other objects in the base document. The advantage is that all individuals' intentions are visualised, thus facilitating the negotiation process. However, there are some disadvantages associated with it. First, the substitution of the base object with the embedded versions will change the context of the object in conflict and may make the document become dirty due to the unpredictable nature of negotiation dynamics. This may cause confusion among the users. For example, in a graphic editor, concurrent "move" operations may result in multiple versions being created and placed in different positions. It is not easy for the users to differentiate between a new object deliberately created by a user and a version resulting from a conflict. Second, multiple versions created may interfere with each other from the point of view of interface design. For example, two concurrent "change colour" operations targeting the same graphic object will result in two versions with different colours being rendered in the same position. Only one of them is visible to the end-user at each site. Finally, the versions created lose their reference point (base object), which may be useful for human users to resolve the conflict.

2.3 Our Approach: Real-Time Alternating Annotation

We observe that during a collaborative editing session, most of the time each user just focuses on his/her own part of work. Nevertheless there are chances when a sub-group of users work together on a sub-task and have intensive discussions. The former is similar to that every participant is involved in a main conference that is held in a main conference room or hall. When discussion or negotiation is necessary, a sub-group of participants can initiate a branch conference parallel to the main one, which may be held in a separate room such that their communications will not interfere with the main track. Only when they reach an agreement will their contributions be presented (integrated into) to the main conference. The main conference (room) and branch conference (room) metaphors are corresponding to main document and branch document in our collaborative editing scenario. The branch document can be seen as a place for a sub-group meeting, only when the users reach a consensus, will it be discarded.

Therefore we introduce a branch document for each object in conflict to prevent the multi-versioning effect from interfering with the main document. All the versions created for one object are presented in one branch document. The bigger the number of objects in conflict, the bigger the number of branch documents that will be created.

Existing multi-versioning techniques support *intrusive negotiation*, whereas our approach can be categorised as *annotative negotiation*. The main document is annotated rather than embedded with branch documents, each of which contains multiple versions of the object in conflict. Annotation is widely used in asynchronous group work. For example, PREP [7] and Quilt [4] are well-known research prototypes supporting annotation. Virtually all commercial document-processing packages, such as, Microsoft Word and Lotus Notes, support some form of annotations. Recently, there are a number of researches on web annotation for supporting group work [1]. In many real-time groupware systems, a voice channel or text chat window is provided to support commenting on the collaborative work in progress. However, the comments (in the form of either voice or text) can only be indirectly integrated into the final document. Furthermore, commenting on an artefact is less efficient and harder to understand than providing its alternative versions. Our approach can be categorised as *real-time alternating annotation* in real-time group editors. It has the advantage that users' contrasting intentions are visualised in alternative versions and their contributions can be directly applied to the document without being wasted. To the best of our knowledge, there are no existing real-time group editors implementing such a mechanism. It works as follows:

As soon as the first conflict occurs, the object in conflict is locked automatically by the system, and the local user is not allowed to further edit it. The system creates two versions from it in a branch document according to the multi-version approach. Any operation targeting the base object, which is received from a remote site, is applied to the object versions in the branch document. Whether the local user is allowed to edit any of them depends on the post-locking scheme implemented [12]. As soon as the lock is synchronised (i.e.,

the local application has known that the object is locked globally), the system searches for the operations that have been applied to the base object but involved in the conflicts. By undoing all these operations in the base object, we can obtain a globally consistent referential object and a set of versions.

It is worth noting that users at different sites may not see the same referential object until the lock on it is synchronised. All the undone operations will not be kept in its history buffer. The base object will be unlocked only when a group consensus (a group intention) has been reached and applied to it.

With the preservation of the referential base object, it is possible to hide the conflicts from those users who are not involved in them. On the one hand they may not be interested in them and may not like to be interfered with. It is enough for them to know that the object is in conflict. On the other hand, the involved users may not want to reveal their privacy (or personal opinions on the conflicts) to those beyond the sub-group. This has a significant impact on the design and implementation of the group user interface.

Various mechanisms are necessary in order to support users in reaching a consensus. However, we assume a consensus or group intention has already been reached in this paper. A group intention consists of a subset of the operations involved in the conflict. The operations contained in a group intention are referred to as *team operations*. We are only interested in how the team operations are applied back to the main document. No matter which negotiation coordination scheme is employed and how long it takes to reach a consensus, the final decision is made either by a representative or by a group of users via a voting protocol. In other words, a group intention is realised by either a representative or the sub-group of users involved in the conflict. If a consensus cannot be reached due to some reasons, the branch document will be kept as an annotation to the base object.

When a group consensus is reached, it seems straightforward to apply the team operations to the main document to realise a group intention. However, there are three special issues we need to consider. First, our conceptual differentiation of group intention from individual intention brings a puzzle to the system. Each individual operation has an owner (originating site), so who is the owner of the team operations? Second, a team operation may originally be contributed by an individual user and was executed according to the causal order. However, other operations causally following it may have been applied to the main document since then. Directly applying the operation to the main document may cause causality violation. Third, the execution of team operations will affect the document state (or context of operation), and thus interfere with other operations. Global consistency may not be maintained without additional techniques to support contextual intention preservation.

All the above three issues are related to how to re-time-stamp the team operations. Obviously, their original state vectors cannot be used, otherwise causality violations may occur. Similarly, if we use the current local state vector, global inconsistency will occur, since different sites may have different local state vector values when executing the same team operation.

3 Representative-Based Time-Stamping Scheme

Unlike individual operations, a team operation is not necessary to be multicast to remote sites where all its information except the new time-stamp is already available. Only the message of the decision result is necessary for those sites. Nevertheless, team operations applied to the chosen version may be recorded in different orders (for concurrent operations) at different sites, so they need to be totally ordered such that global consistency can be guaranteed when a representative applies them to the main document.

As soon as the representative makes the final decision by choosing a version, the decision message is time-stamped and multicast to all participating sites. Before executing any remote operation, the local application at the representative's site re-time-stamps the team operations according to the scheme in Definition 3 as if they were newly generated by the local user. Similarly, as soon as the decision message is received and executed at a remote site, the team operations are re-time-stamped and executed there consecutively. Their time-stamps are based on the one of the decision message, which is independent of the state vector at the receiving site.

Definition 6. Representative-based time-stamping scheme of the team operations*: Suppose the identifier of the representative site is k and the decision message is time-stamped by* SV_0*. The state vectors of the list of totally ordered team operations* $[O_{T_1}, \ldots, O_{T_t}, \ldots, O_{T_n}]$ *are specified as follows:* $SV_t[k] := SV_0[k] + t$*,* $SV_t[i] := SV_0[i]$*, for* $t = 1, \ldots, n$*;* $i = 1, \ldots, N$*, and* $i \neq k$*, where* N *is the number of sites.*

Accordingly, the virtual clock (state vector) maintained at each site needs to be updated when executing the team operations.

It is worth noting that the time-stamping of the decision message and team operations must be performed atomically at the representative site, otherwise globally inconsistent time-stamping results will occur, and this will lead to inconsistent execution effects. Similarly, the decision message and team operations must be executed atomically at each site.

Apparently, with the above scheme, a group intention becomes an individual intention (of the representative) after its application to the main document.

4 Temporal Relations of Team Operations

If the voting technique is employed to resolve a conflict, the voters are of equal status. The voting result is a "true" group intention rather than the representative's intention. Nevertheless, we need to introduce a "virtual site" (or "virtual user") to represent the "group". Accordingly, the state vector needs to be extended to incorporate the extra site. Based on this extended virtual clock, it is clear that all the team operations in a group intention can be considered as ones "generated" by the virtual user at the virtual site. Therefore, the temporal relationship between an individual operation and a team operation can be

determined in the same way as that between two individual operations. The operations generated by the same human user are ordered according to the local causal relationship (physical clock). However, the way in which the team operations associated with different branch documents are ordered is unknown at the moment.

Consider a document consisting of independent objects. Since the team operations involved in the same branch document will be applied to the base object in the same order (total order) at each site and team operations from different branch documents will not interfere with each other, the order of their execution is irrelevant to the final rendering effect. In other words, team operations from different branch documents are contextually independent.

However, with a plain text document where characters are sequentially ordered [14], inconsistency will occur if two or more group intentions (resulting from different branch documents) are agreed-upon and directly applied to the main document in different orders at different sites. It is also uncertain how to reconstruct the generation context (i.e., the document state on which an operation is generated) of a newly arrived remote operation without the knowledge of group intention realisations at its originating site.

For example, assume O_1 is generated at site i after a group intention (consisting of an operation O_{T_1}) has been realised. When O_1 arrives at site j, where O_{T_1} has not been realised, but another team operation O_{T_2} has been applied to the main document. In this scenario, it is not possible for site j to determine O_1's generation context with the time-stamping scheme defined in Section 2.

From the perspective of group undo, all the team operations must be properly ordered such that a "group" player (i.e., the user who has obtained a special permission from the group to play the role of the "group") can undo the team operations in a globally consistent way. Therefore, a new ordering scheme is necessary for the team operations. The major challenge is to find temporal information characterising the group intention of a branch document, which can be used to order different group intentions of different branch documents. If the team operations are time-stamped in a globally consistent way, then the preservation of group intentions is similar to that of individual intentions.

We assume a group consensus is reached only after a site has received a voting message from each of the involved users. Obviously, both the totally oldest and youngest voting messages are globally unique. They can be used to characterise the group intention of a branch document. Group intentions of different branch documents then can be ordered according to the total order of their respective totally oldest (or youngest) voting messages. However, since voting may take some time to complete (due to, for instance, hesitation of some of the users), a branch document with an older totally oldest voting message may reach a consensus later than the one with a younger totally oldest voting message. The team operations resulting from the latter will not be allowed to apply to the main document until those of the former have been applied. In other words, the group intention of the latter may be blocked from execution for a long time. This seems abnormal to the end users. Therefore, we now discuss the application of

the temporal information of the totally youngest voting message to the ordering of team operations.

We assume that the set of team operations resulting from a branch document is associated with the state vector of its totally youngest voting message. Formally, a group intention can be represented by a tuple: $\langle\{O_1, O_2, \ldots, O_m\}, SV\rangle$, where $\{O_1, O_2, \ldots, O_m\}$ is the set of team operations from a branch document, and SV is the state vector of the totally youngest voting message associated with it. In order to be consistent with the representation of individual operations, we use an operation vector $\mathbf{O}$ to denote the set of team operations, which share the same time-stamp before being applied to the main document. Obviously, the intention of an operation vector is a group intention. Nevertheless, each operation in the vector will have a distinct time-stamp when they are realised. Therefore, the causal ordering and total ordering relations defined in Section 2 can be easily extended without major modifications (Since each voting message has a unique state vector, we will not differentiate a voting message from its state vector for convenience in the sequel).

Definition 7. Total ordering relation of group intentions*: Given any two sets of team operations* $\mathbf{O}_I$ *and* $\mathbf{O}_J$ *at the same site,* O_i *and* O_j *are the totally youngest voting messages associated with them respectively.* $\mathbf{O}_I$ *is said totally preceding* $\mathbf{O}_J$*, expressed as* $\mathbf{O}_I \Rightarrow \mathbf{O}_J$*, if* $O_i \Rightarrow O_j$*.*

Definition 8. Virtual ordering relation of team operations*: Given any two team operations,* O_a *and* O_b*.* O_a *is said virtually preceding* O_b*, expressed as* $O_a \Rightarrow_V O_b$*, if (1)* $O_a \in \mathbf{O}_I$*,* $O_b \in \mathbf{O}_J$*, and* $\mathbf{O}_I \Rightarrow \mathbf{O}_J$*,* $(I \not= J)$*; or (2)* $O_a, O_b \in \mathbf{O}_I$*, and* $O_a \Rightarrow O_b$*. Where* $\mathbf{O}_I$ *and* $\mathbf{O}_J$ *are two sets of team operations.*

Generally, the virtual ordering relation $\Rightarrow_V$ is consistent with neither the total ordering relation $\Rightarrow$ (defined for individual operations) nor the causal ordering relation $\rightarrow$. This will not be a problem, since the virtual ordering relation is defined on the old time-stamps of the team operations. Before a team operation is applied to the main document, it will be re-time-stamped such that individual operations and team operations can be treated in the same way. The virtual ordering relation is introduced to artificially order those operations that involve in conflicts. It is also a total ordering.

An individual operation is first generated by a real site without a time-stamp associated with it. If the system is not in the process of executing a remote operation, the operation will be time-stamped in terms of the local virtual clock (state vector) and executed immediately at the local site, meanwhile the system will update the clock. The time-stamping, executing, and clock updating must be completed atomically. The newly generated operation can be executed immediately since it is always causally ready. In contrast, a team operation is "generated" with an old time-stamp assigned by its originating site, as soon as a consensus is reached. Like the concept of relative time, consensus is also a local perception of global states. Therefore, a team operation will be "generated" once at each of the participating sites. We can assume it is "generated" by the virtual

user at the virtual site. The problem is when to execute it and how to re-time-stamp it. The individual operations generated by a real site is ordered according to its physical clock, naturally, the team operations will be ordered in terms of the virtual ordering relation. Therefore, we arrive at the following condition for executing team operations:

Definition 9. Virtually ready*: A team operation is said to be* virtually ready *for execution at a site only if it is virtually preceding all unrealised (or not yet applied to the main document) team operations.*

5 Group-Based Time-Stamping Scheme

Although the team operations "generated" already have their old time-stamps, they must be re-time-stamped before they can be applied to the main document. With the introduction of the virtual site (user), the existing real site-based time-stamping scheme can be extended such that the team operations can be properly time-stamped.

Definition 10. Group-based time-stamping scheme*: Assume each site maintains a state vector SV with* $N+1$ *components. Initially* $SV[i] := 0$*, for all* $i \in \{1, \ldots, N, N+1\}$*. After executing an operation generated at site* i*,* $SV[i] := SV[i] + 1$*, where site* i *can be a remote site, the local site, or the virtual site* $(N+1)$ *(if it is a team operation). Whenever an operation is generated at a real site* k $(k \neq N+1)$*, it is first time-stamped with a state vector* SV_r *and then executed at the local site immediately and multicast to remote sites.* $SV_r[k] := SV[k] + 1$*;* $SV_r[i] := SV[i]$ $(i \in \{1, \ldots, N+1\}$ *and* $i \neq k)$*; where SV is the current state vector of site* k*. When a team operation becomes "virtually ready" at real site* k*, it is re-time-stamped with* SV_v *and then executed at the local site without being multicast to remote sites.* $SV_v[N+1] := SV[N+1] + 1$*;* $SV_v[i] := SV_{\max}[i]$ $(i \in \{1, \ldots, N\})$*, where* $SV_{\max}$ *is the state vector of the totally youngest voting message associated with the team operation.*

In the example presented in Section 4, because independent group intentions must be time-stamped and executed according to the virtual order, if $O_{T_1} \Rightarrow_V O_{T_2}$, then O_{T_2} cannot be executed before O_{T_1}. When executing O_1 at site j, O_{T_1} must have been executed because the re-time-stamped O_{T_1} is causally preceding O_1. So, the execution order at site j in the scenario is not possible with our new time-stamping scheme. In addition, since O_{T_2} has not been executed before O_1 is generated, O_1 is concurrent with the re-time-stamped O_{T_2}.

Obviously, the time-stamp of a team operation depends on the number of team operations having been executed at the real site and the state vector of its totally youngest voting message. Since the latter is globally unique, if the team operations are globally ordered (i.e., executed only when they are virtually ready), the global consistency of the time-stamping scheme can be guaranteed.

When a consensus is reached, the team operations associated with it may not necessarily be virtually ready, since there may exist multiple branch documents

reaching a consensus concurrently. To put it in another way, the totally youngest voting messages of different branch documents may be concurrent with each other. Although our virtual ordering relation is based on the total ordering one, the system cannot know whether the totally youngest voting message associated with a consensus is the virtually oldest one until it is synchronised (i.e., all operations concurrent with it have been executed).

Therefore, the team operations of a group intention can be re-time-stamped and applied to the main document only when the totally youngest voting message associated with it is both globally synchronised and virtually (or totally) the oldest (among those of the currently unrealised group intentions). Obviously, the execution of a team operation is not as responsive as that of an individual operation.

It is desirable that a system supports group intention preservation by both representation and voting. However, the time-stamping schemes proposed for them are not consistent. Therefore one should be adapted to the other. Obviously, the group-based scheme is more general than the representative-based one, so the latter must be extended. The price we have to pay is the introduction of one extra dimension for the virtual site in the state vector. More importantly, group intention preservation by representative can be considered as a special case of the one by voting, i.e., only the representative votes. The decision message generated by the representative is a voting message. It is both the totally oldest and youngest. As soon as the representative makes a decision, the consensus is reached, i.e., there is no voting time. The operations contained in the chosen version become team operations "generated" by the virtual user instead of the representative. Since there may exist multiple concurrent decisions made by multiple representatives on different branch documents, the group consensus cannot be applied to the main document until its decision message is globally synchronised and virtually preceding all the other decision messages. Therefore, unlike the representative-based scheme, the group-based scheme is not responsive with regard to the execution of team operations. Nevertheless this is acceptable, since users may have been involved in the negotiation process for quite a period of time, an extra tiny delay will not be a serious problem.

6 Conclusions

This paper started with an examination of the problems of the existing multi-versioning techniques in supporting negotiation. They are "intrusive" in the sense that the embedded versions interfere with the other objects contained in the document in terms of not only data model but also data view. To avoid the interference, we introduced a pair of metaphors, i.e., main conference and branch conference, to describe the two different circumstances of collaboration, which can be referred to as *consensus-based cooperation* among all the members of the group and *conflict-driven negotiation* among a subset of the members of the group. The cooperation process in the former case is generally smoother, due to the assumption that conflicts are rare, whereas the negotiation process

in the latter case involves many interactions with contrasting intentions among the members of the sub-group.

The introduction of the metaphors leads to our new approach, categorised as annotative negotiation, where the main document is annotated rather than embedded with branch documents. This approach has the advantage that users' contrasting intentions are visualised in alternative versions and their agreed-upon contributions can be applied to the document without being wasted. However, this raises a new technical issue of group intention preservation, i.e., how to integrate the consensus into the main document. Directly applying group intentions to the main document may lead to intention violations. The technical solution to this problem lies in re-time-stamping the team operations contained in the group intentions such that they can be executed in the same way as individual operations are executed. To this end, we proposed the representative-based and group-based time-stamping schemes. The latter is applicable to both the representative and group voting decision mechanisms. Conceptually, the concepts of virtual user and virtual site are introduced and associated to group intentions; technically, a virtual ordering relation is defined to totally order team operations. Consequently, individual intentions and group intentions can be treated in a consistent manner. The scheme has been implemented in our research prototype called POLO [13].

Acknowledgements. The work presented in this article is partly sponsored by an Australian Research Council (ARC) grant. The work of Liyin Xue was also supported by an award from Macquarie University.

References

1. J.J. Cadiz, A. Gupta, J. Grudin. Using web annotation for asynchronous collaboration around documents. In Proceedings of ACM Conference on Computer Supported Cooperative Work, Dec. 2000, pp. 309–318.
2. C.A. Ellis and S.J. Gibbs. Concurrency control in groupware systems. In Proc. of ACM SIGMOD Conference on Management of Data, May 1989, pp. 399–407.
3. L. Lamport. Time, clock, and the ordering of events in a distributed system. In CACM 21(7), July 1978, pp. 558–565.
4. M.D.P. Leland, R.S. Fish, R.E. Kraut. Collaborative document production using Quilt. In Proc. ACM Conference on Computer Supported Cooperative Work, 1988, pp. 206–215.
5. T. P. Moran, K. McCall, B. van Melle, E. R. Pedersen, and F.G.H. Halasz. Some design principles for sharing in Tivoli, a white-board meeting support tool. In S. Greenberg, S. Hayne, and R. Rada (eds.), Groupware for Real-time Drawing: A Designer's guide, McGraw-Hill, 1995, pp. 24–36.
6. J. P. Munson and P. Dewan. A concurrency control framework for collaborative systems. In Proceedings of ACM CSCW 1996, pp. 278–287.
7. C. M. Neuwirth, D. S. Kaufer, R. Chandhok, and J. H. Morris. Issues in the design of computer support for co-authoring and commenting. In Proceedings of the ACM Conference on Computer Supported Cooperative Work, Nov. 1990, pp. 183–193.

8. A. Prakash. Group editors. In M. Beaudouin-Lafon (ed.), Computer Supported Cooperative Work, John Wiley & Sons, 1999, pp. 103–133.
9. M. Raynal and M. Singhal. Logical time: capturing causality in distributed systems. In IEEE Computer Magazine, 29(2), Feb. 1996, pp. 49–56.
10. C. Sun and D. Chen. A multi-version approach to conflict resolution in distribute groupware systems. In Proceedings of International Conference on Distributed Computing Systems, April 2000.
11. C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. In ACM Transactions on Computer-Human Interaction, 5(1), March 1998, pp. 63–108.
12. L. Xue, K. Zhang, and C. Sun. Conflict control locking in distributed cooperative graphics editors. In Proceedings of the 1st International Conference on Web Information Systems Engineering (WISE 2000), Hong Kong, IEEE CS Press, June 2000, pp. 401–408.
13. L. Xue, K. Zhang, M. Orgun, and C. Sun. Version composition and identification schemes in distributed real-time groupware systems. In Macquarie Computing Reports, No. C/TR01-01, Macquarie University, February 2001.
14. L. Xue, K. Zhang, and C. Sun. An integrated post-locking, multi-versioning, and transformation scheme for consistency maintenance in real-time group editors. In Proc. of the 5th Intentional Symposium on Autonomous Decentralised Systems, Dallas, Texas, USA, IEEE CS Press, Mar. 2001.

A User-Centred Consistency Model in Real-Time Collaborative Editing Systems

Liyin Xue[1], Mehmet Orgun[1], and Kang Zhang[2]

[1] Dept of Computing, Macquarie University
Sydney, NSW 2109 Australia
{lyxue, mehmet}@ics.mq.edu.au
[2] Dept of Computer Science, University of Texas at Dallas
Richardson, TX 75083 USA
kzhang@utdallas.edu

Abstract. Consistency maintenance is an important issue in groupware systems. The existing consistency models are system-oriented and focus on objective and static aspects of consistency without sufficiently taking into account the involvement of human users. It is necessary to investigate how users perceive consistency and how systems can support the users in realising their expectation. This paper addresses consistency from the point of view of multi-user interaction and proposes a new user-centred consistency model. It argues that consistency is an agreement (or inter-subjectivity) among human users in groupware systems. Such a user-centred interpretation leads to the introduction and differentiation of externalised consistency and emergent consistency. The latter captures the fundamental characteristic of collaborative work, where human users' intentions and negotiation play a crucial role. The paper examines the constituent parts of the model in detail.

1 Introduction

Distributed real-time collaborative editing systems support multiple users to view and edit the same document concurrently from geographically dispersed sites connected by a communication network, such as the Internet [5,13]. They reduce the time required to complete a task by avoiding the overhead of merging different versions of the same document into one.

With the development of the Internet, online wide area collaboration has become popular. Currently the Internet has a rather serious restriction of bandwidth, which will incur large communication latency. However, the fundamental challenge to groupware researchers is not the limited bandwidth available but the constant speed of light. Inter-continental communication delay cannot be reduced to being lower than the physical limit. This has a significant impact on the design of groupware systems. First, with the centralised systems, users cannot see their own actions immediately, since the messages have to go through a round trip. Therefore, local commands (including lock requests) are not of responsiveness. Second, remote users cannot see immediately the actions that other users

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 138–150, 2002.

have performed. The WYSIWIS (what you see is what I see) property no longer holds.

The responsiveness of local commands can be maintained if the objects to which users' actions are applied are replicated at all participating sites [2]. However, replication results in a serious consistency maintenance issue due to the possibility of concurrent editing of the same object by multiple users.

Technically, groupware systems have inherited many consistency maintenance mechanisms, such as, locking and transaction, from traditional database systems, where the basic principle is conflict prevention. Generally, this is in contradiction to the spirit of collaboration. Since users are accustomed to the working style associated with single-user editors, unconstrained and non-restrictive editing is highly desirable.

Theoretically, conflict is inevitable in collaborative work. Conflict may not necessarily be destructive to the collaboration process. Instead, it contributes to the emergence of group intentions. For example, with collaborative authoring on a document, even though it is possible to allocate each author with a concrete task, it is not possible for the organiser to have everything in mind beforehand. Discussion and negotiation play a major role in the formation of the document. Even a labour division scheme emerges from group discussion and negotiation. Therefore, conflict control and negotiation support are important issues in the design of groupware systems [17].

Due to the above considerations, we advocate a shift of consistency maintenance strategies from *conflict prevention* to *conflict control* (cure or management), and a shift of consistency maintenance mechanisms from *system-based automatic conflict resolution* to *user-centred negotiation support*. Our major concern is the intentions of human users rather than the pre-determined behaviour of the systems. We observe that the existing research work has not sufficiently taken into account human users' involvement in the evolution of consistent documents. Therefore it is necessary to investigate how users perceive consistency and how systems can support the users in realising their expectation. This calls for a new user-centred consistency model.

The rest of the paper is organised as follows. We start with an examination of related work in Section 2. We then analyse consistency from system's and user's perspectives in Sections 3 and 4 respectively. In Section 5, an analytical framework is proposed to examine various meanings of human user intentions. Major new research issues in consistency maintenance are outlined in Section 6. Finally, Section 7 summarizes the major contribution of this paper.

2 Related Work

Consistency maintenance is a well-studied topic in parallel and distributed systems, conventional database systems, and groupware systems. Various consistency models have been proposed. They are system-oriented and focus on objective and static aspects of consistency without taking into account the involvement of human users. Generally speaking, there is little explicit work on the

advancement of consistency models in groupware systems. Many research prototypes are based on existing models available from related areas. Generally, they are either too restrictive or unable to provide support for the situated negotiation among participating users.

There are groupware systems that implement the transaction mechanism, based on the serialisable consistency model from database systems. They reduce the consistency problem to that of testing for serialisable schedules [6,11]. The serialisation approach implemented in GroupDesign [9] and LICRA [8] is based on the sequential consistency model from distributed shared memory systems, which guarantees that operations generated at each site are executed in a global order at all participating sites.

In distributed shared memory systems, *a consistency model is essentially a contract between the software and the memory.* If the software agrees to obey certain rules, the memory promises to work correctly. If the software violates these rules, the correctness of memory operation is no longer guaranteed [1]. Many consistency models trade consistency for better performance or ease of programming. A parallel view on consistency in groupware systems has been proposed by Dourish [3]. He introduced the notion of *consistency guarantees* as a technique to increase the effectiveness of the explicit semantics approach. Consistency guarantees regard various locks as guarantees of some level of achievable consistency, i.e., if a client promises to follow the restrictive rules determined by the server, it receives some guarantee of future consistency. In other words, *consistency is a contract between the client and the server.* However, a client can break a promise, in which case the server is no longer held to its guarantee. This has a significant implication for the design of a groupware toolkit to provide support for opportunistic working styles.

The only consistency model that takes user intentions into account was proposed by Sun et al [15,16]. However, it is still system-oriented. We shift our focus from system to user, and differentiate between user's perspective and system's perspective on consistency. We address consistency from the point of view of multi-user interaction, and argue that *consistency is an agreement* (*or intersubjectivity*) *among human users* in groupware systems. Such a user-centred interpretation results in the introduction and differentiation of externalised consistency and emergent consistency. The latter captures the fundamental characteristic of collaborative work, where human users' situated intentions and negotiation play a major role.

3 System's Perspective on Consistency

There are two aspects of system's perspective on consistency. First, a system specifies the ways (e.g., timing, order, etc.) in which multiple processes or users can jointly access to the shared data. For example, only serialisable transactions are allowed in database systems. This is the traditional concurrency control (conflict prevention) issue in database systems. Second, the system specifies various

constraint rules on data items. This is the integrity constraint issue in database systems.

However, not all constraints can be specified in advance in multi-user environments. On the one hand, consistency may become contingent on the negotiation process involving multiple users. On the other hand, some consistent state of the data may not satisfy all of the involved users, that is, the perspectives of individual users and the group on consistency are different. It is generally not possible for the system alone to automatically maintain the consistency required by human users. Therefore, it is necessary to distinguish *individual's perspective* from *group perspective* on consistency, and what we are concerned with is not *objective consistency* but *subjective consistency*. However, some part of the subjective perspectives must be *externalised* such that consistency maintenance mechanisms can be provided with the systems in advance. The designers of groupware systems must anticipate the mutually acceptable perspectives (or *inter-subjectivity*) of multiple users, which can be predefined. This static inter-subjectivity reflects only a rather limited aspect of consistency. A dynamic version of inter-subjectivity can *emerge* only after a negotiation process during the real application of the groupware system. Therefore, we introduce two notions of consistency, namely, externalised consistency (or *a priori* consistency) and emergent consistency (or *a posteriori* consistency).

3.1 Externalised Dimension of Consistency

In a replicated environment, since each user can concurrently view and modify her/his respective copy of the document under editing, final documents would not be identical among participating sites if the system were not provided with a proper consistency mechanism. It is generally required that all copies of the shared document be identical in terms of *model* at all sites at quiescence, i.e., all operations generated by the users have been executed at all sites. This consistency criterion is called *convergence property* [4,16]. *View* synchronisation is important when multiple users are involved in a real-time discussion and negotiation process.

The convergence property reflects the relationships among multiple replicas of the same document. It is about *inter-copy consistency*. Besides this, the data of each copy may be required to be consistent (*intra-copy consistency*) in various linguistic levels in many applications. For example, a multi-user integrated development environment may support multiple programming languages. It can be configured such that it provides a check of the correctness of user inputs in terms of lexical constructs of a language. This is the issue of *lexical consistency*. The syntax of a programming language can be specified by a grammar as in most of the current commercial programming languages. A correct program must satisfy the syntax specified, that is, *syntactic consistency* must be maintained. Similarly, a document may be required to be formatted and organised in a specific way, or *structural consistency* property must be maintained. *Semantic consistency* is important but difficult to specify in advance.

3.2 Emergent Dimension of Consistency

The extent of externalisation of group perspectives on consistency is rather limited in the sense that it reflects only a part of the rich connotation of subjective consistency. For example, document semantics is generally difficult to specify in natural language environments. Consistency is not only application and user dependent, but also context dependent, where multiple users are involved in a situated conversation process.

A correctly spelt English word may not be suitable for a particular semantic context. Co-authors can negotiate with each other to find a proper synonym. This is a simple example of emergent consistency.

Suppose a shared document contains a sentence:

A: "John forgot lock the door."

There is an English grammar error in it, which may be detected by a well-designed system. However, the best the system can do is to notify the users and present the following two alternatives:

B: "John forgot to lock the door."

C: "John forgot locking the door."

Both B and C are syntactically correct, but with different meanings. The involved users need to discuss their appropriateness for the semantic context [19]. This is an example of emergent semantic consistency.

The system can enforce externalised consistency automatically through either a fixed implementation or run-time configuration. Generally, a system does not have sufficient knowledge of what consensus the users are to reach in advance. Therefore, the maintenance of emergent consistency can only be supported rather than enforced by groupware systems.

4 User's Perspective on Consistency

In database systems, since all transactions are atomic and considered independent of each other, the semantics of a transaction is enforced as long as it commits. However, in groupware systems, it is not acceptable to exclude other users from viewing or editing the shared data for a long period of time. Generally intermediate results of transactions are opened up for multiple users such that they are no longer serialisable [5,7].

In groupware systems, the operation issued by a user reflects her/his perspective on the current document state (i.e. context of the operation) and his intention to change it to a new state. Since the context may be changed by concurrent operations issued by other users, the result of the execution of the original operation on the new context may not be the user's original intention. Therefore intention preservation becomes an important issue of consistency maintenance in collaboration systems. However, most of the earlier prototypes of group editors employ various locking techniques as concurrency control mechanisms, which are similar to the database transaction approach, thus no intention preservation problem needs to be considered.

In groupware systems, a particular intention violation problem was first solved by operational transformation approach in the well-known *Grove* collaborative outline editor [4]. Due to the concurrent execution of operations at different sites, the context or document state on which an operation was generated may be changed by other operations at other sites. The operation must be properly compensated in order to be executed in a new context. The compensation functions are called transformation functions. None of the operations need to be aborted. Later researchers considered intention preservation as an important issue in groupware systems [13,15,16]. Some progress has been made since then [12,14,17,18,19]. However, the nature and significance of intention preservation have not yet been fully investigated. For example, intention preservation is simply juxtaposed with convergence property in Sun et al's consistency model [16]. The problem is obvious, since the preservation of multiple users' concurrent conflicting intentions may make document convergent but semantically inconsistent. For example, the embedded multiple versions for the preservation of conflicting intentions will interfere with the other objects of the document [10, 14,20].

From the system-user dichotomy, we can easily identify the place of intention preservation in consistency maintenance. It is both a consistency property and maintenance mechanism based on the user's perspective on consistency. Users are the final judgers of consistency. Consistency is an agreement among the involved users. Therefore, the concept of intention preservation could be seen as general as consistency maintenance if we would extend the meaning of intention. Nevertheless, we reserve the latter to cover both the system's and user's perspectives on consistency.

5 An Analytical Framework for Intentions

Similar to the distinction of individual perspective and group perspective on consistency, we need to differentiate between the concepts of *individual intention* and *group intention*. An individual user's intention is usually represented by one or more operations generated by the user (we use user's intention and intention of an operation interchangeably in this paper). The intention of one user may conflict with that of another user. A group intention represents the one that is agreed upon among the group members. In other words, a group intention is the one resulting from merging individuals' intentions via a synergistic coordination or negotiation process among the involved users. It can be categorized into *externalised group intention* and *emergent group intention*.

5.1 Individual Intentions

In terms of the type of user actions, individual intentions can be categorised into *editing intention*, *annotating intention*, and *negotiating intention*, etc. Here we focus on the analysis of editing intentions.

Contextual Intentions: Whenever a user is to issue an action, she/he must have been aware of the current context or document state and a decision or intention has been in her/his mind. Therefore, individual users' intentions are generally context dependent or *context sensitive*. Any context sensitive intention must be executed on its original context, or compensated (transformed) such that it can be executed on a new context, otherwise intention violation will occur. In contrast, *context free* intentions can be performed in any context. In fact, the pessimistic locking approach implemented in collaborative editors assumes any intention is absolutely context sensitive and concurrent modifications of the same part of a document are not allowed. However, the scope of a context is generally rather limited. Modifications beyond this scope may not be inter-related with it. The granularity of context is a very important issue in the specification of consistency maintenance policies.

Connotative Intentions: In database systems, any user or application can only see the consistent states of data. In other words, a database can only move from one consistent state to another from the viewpoint of users (or applications). Nevertheless, during the editing process, users are allowed to see inconsistent states of the document in collaborative systems. They may perform some actions on it to make it consistent, while inconsistency may persist for some time during the editing process. In other words, whenever issued, an action is connoted with rich indirect intentions of its issuer, in addition to its direct intention.

For example, consider the example in Section 3.2, a user may issue an operation O_1 to insert "to" at the position between "forgot" and "lock", thus changing sentence A to B. The system can mechanically interpret the user's *explicit intention* reflected in the operation. However, it does not comprehend her/his *implicit syntactic intention* (e.g., correcting a syntactic error) and *semantic intentions* (e.g., the meaning of sentence B instead of C) connoted with the operation.

Connotative dimension of intention is important when users need to explicitly discuss or negotiate about the document content to which an operation has been applied. Neglecting such a factor may cause intention violation. An extra communication channel, such as, a voice channel, may be necessary for communicating the implicit syntactic and semantic intentions.

Transactional Intentions: A user may issue a group of interdependent operations, like those contained in a transaction in database systems. Other users may be allowed to read the partial execution results but not to issue any operation on the same objects targeted by those operations, such that the user's intention is not interfered with. The maintenance of atomicity of intention consisting of multiple sub-intentions, which we call *transactional intention*, is an important issue in real-time collaborative editing systems.

5.2 Externalised Group Intentions

It would be tedious for users to negotiate with others about every intention. Only when the users feel that it is necessary to have a discussion or when the system

automatically detects that there is a (potential) conflict, should the users be brought together to participate in a negotiation process. An individual intention committed without negotiation is in fact an externalised group intention, which had been agreed upon even before the users started editing the document, that is, an operation that does not intentionally conflict with other operations can be applied to the document directly without negotiation by default. An externalised group intention can be an individual intention or a combination of multiple individual intentions based on predefined rules. All involved users must be aware of these rules. If they disagree with each other on them, the system must be reconfigured to adapt to the new consensus.

Since group intentions generally involve multiple individual intentions, they can be embodied in the relationships between individual intentions (or operations). Moreover, an operation is defined on a particular context, which is the result of the execution of multiple operations and may be concurrently modified by other operations. Therefore the discussion of contextuality always resorts to the relationships of operations. Two operations may have multiple relationships in terms of their execution (or generation) order, the targeted objects, and the object attributes to be changed.

Temporally and intentionally dependent relations: If operation O_1 is causally preceding operation O_2, O_2 is said *temporally dependent* on O_1. However temporal dependency may not necessarily mean *intentional dependency*. For example, if O_1 is not in the scope of O_2's generation context, O_2 is intentionally independent of O_1. Therefore, given two operations, if one is temporally dependent on but intentionally independent of another, they can be executed in any order, i.e., they are commutative. In this case, it is not necessary to maintain the causality-preservation property (i.e., all operations are executed in the causal order) presented in Sun et al's consistency model [16]. However this case is too special. Intentional dependency may cause troubles when an operation is undone. Its dependent operations must also be undone. It is hardly possible for the system to know the relationships automatically. However, operations targeting semantically related data may be controlled by integrity constraints specified in advance.

Semantically compatible and conflict relations: If two operations target syntactically and semantically unrelated regions of a document, they are said to be semantically compatible, otherwise they are semantically conflicting with each other. For example, two operations targeting the same object and changing the same attribute to different values are semantically conflicting with each other. It is application and user dependent whether two operations targeting the same region or object of a document are conflicting or compatible. Semantic relationships are very important in configuring the consistency maintenance policies.

Intentionally compatible and conflict relations: Any two operations that are both concurrent and semantically conflicting are said to be intentionally

conflicting. Otherwise they may be intentionally compatible. Obviously if intentionally conflict operations are applied to the document together, intention violations will occur, whereas intentionally compatible operations can be applied to the document without interfering with each other. However, they can be defined in various ways depending on the agreed upon perspectives of the users. For example, two operations targeting the same paragraph may be defined as conflict and explicit negotiation for a group intention is necessary [19]. They may also be treated as compatible and a group intention is reached automatically via a proper transformation of them [4].

5.3 Emergent Group Intentions

Operations targeting completely different parts of a document are in fact not synergistic, though all contribute to the formation of the document. Only when they target the same part or semantically related parts, synergy may happen. It is generally infeasible for the system to know the negotiation result, i.e., emergent group intention, in advance.

Consider the example presented in Section 3.2 again. Assume user i issues an operation O_1 to insert "to" at the position between "forgot" and "lock", thus changing sentence A to B. User j concurrently issues an operation O_2 to insert "ing" at the ending position of "lock", thus changing A to C.

The existing operational transformation schemes (in fact, externalised rules for group intention) can preserve the direct intentions of the two operations such that the final document contains sentence D.

D: "John forgot to locking the door."

Apparently this result is still syntactically incorrect, though both B and C are syntactically correct. Neither of the users' connotative intentions as in sentences B and C is preserved. In addition, the meanings of sentences B and C are different. It is not possible for the system to accommodate semantically conflicting intentions into one correct sentence.

From the above example we see that an externalised, pre-specified rule for group intention should be adaptable to various situations. The rule should be specified such that two operations targeting the same sentence or paragraph are conflicting with each other in this example. The only way to preserve the above connotative intentions of the users is by annotation or multi-version. After negotiation, both of the users may agree on one of the versions, i.e., either sentence B or C. For more complicated cases, users may negotiate with each other over multiple loops by modifying their respective versions and merging them such that a group intention will eventually emerge.

6 The Implication of the User-Centred Consistency Model

According to the above discussion, we illustrate the user-centred consistency model in Figure 1. Emergent consistency and emergent group intention must be consistent with their respective externalised ones.

From the user-centred consistency model, we know that a consistency maintenance scheme must deal with both externalised consistency and emergent consistency. A system may be required to preserve various individual intentions, externalised group intentions, and emergent group intentions. The system maintains the externalised consistency and preserves externalised group intentions automatically without the involvement of cooperating users.

Intention preservation plays a crucial role in supporting the maintenance of consistency. If users are not informed about each other's real intentions, they simply cannot negotiate. Furthermore, facilitation mechanisms are necessary to support users in obtaining emergent group intentions. Therefore the focus of consistency maintenance shifts from traditional conflict prevention to *intention preservation* and *negotiation support* in unconstrained real-time distributed collaborative editing environments. This raises new research issues and calls for new mechanisms. We address them from the following three aspects, i.e., the preservation of individuals' intentions, the facilitation of negotiation processes, and the preservation of emergent group intentions.

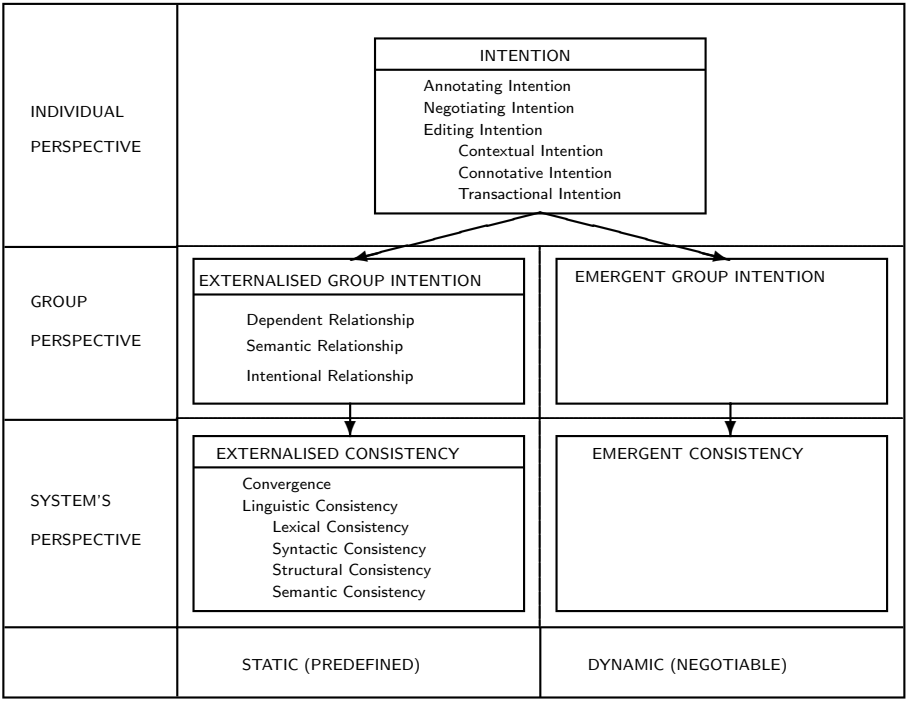


Fig. 1. User-centred consistency model

Preservation of individuals' intentions: The challenge of preserving individuals' intentions arises from the existence of interference of one intention with another. The preservation of any one of the contextual, connotative, and transactional intentions usually resorts to either isolation or compensation of the interference. Currently, there are two approaches to the preservation of contextual intention in unconstrained real-time collaborative editing environments, namely, operational transformation and multi-versioning. The former transforms an operation such that it can be applied to the new context without changing its original intention [4,15]. The latter reconstructs the original context by creating a new version for the operation whose context has been interfered with. The multi-version approach has been proposed to preserve individual's concurrent conflicting intentions in collaborative editors, such as GRACE [14], Tivoli [10], and POLO [18]. However, the schemes presented in Tivoli and GRACE are only conditionally correct [18]. It is not easy to construct a general algorithm for automatic version composition and identification in an unconstrained and highly concurrent collaborative environment.

Facilitation of negotiation processes: There is little work on negotiation support in distributed real-time collaborative editing systems. Its major challenges cover several aspects. First, it is necessary to harness the complexity caused by unconstrained concurrent editing. If the multi-version approach is employed, novel mechanisms are necessary to control the proliferation of versions and facilitate the negotiation process among multiple users. For example, post-locking approach is such a mechanism [17]. Second, coordination mechanisms and multi-modal communication modes for negotiation processes should be investigated. Third, generally, designing a multi-user interface is very difficult. The interface design for negotiation support and conflict resolution is really a challenging issue. Finally, in order to evaluate the current techniques and guide the future research, we need empirical evidence on how people collaborate to resolve conflicts in distributed real-time collaborative environments.

Preservation of emergent group intentions: How to reach an emergent group intention is the problem of negotiation facilitation. If all those individuals' intentions (including negotiating intentions) in conflict are directly applied to the main document, some of them, which the group considers inappropriate, need to be undone or compensated during the negotiation process. With such a scheme, the final emergent group intention has already been realised in the main document when the negotiation ends, and there is no such an issue of emergent group intention preservation. However, the conflict resolution process will interfere with the normal collaboration process and the users may feel being restricted by the "formal" environment. We have proposed an annotating approach, that is, all conflicts are removed from the *main document* and will be resolved in a *branch document* in an "informal" environment, which is an annotation to the main document [20]. Therefore, whenever a group consensus is reached, it must be explicitly applied back to the main document like an in-

dividual intention. Technical mechanisms are necessary for the preservation of emergent group intentions.

7 Conclusion

Considering the crucial role of human users' involvement in groupware systems, we shift the investigation focus from the system to the users, and propose a new user-centred consistency model that emphasises the importance of subjective or negotiating aspect of consistency and human user intentions. Our model can accommodate the issues of intention preservation and negotiation support systematically. We examine the constituent parts of the model, particularly, various aspects of user intentions in detail. Based on the model, a number of major new research issues in real-time collaborative editing systems are outlined, which are categorised into the following three aspects: the preservation of individual intentions, the facilitation of negotiation processes, and the preservation of emergent group intentions.

Acknowledgements. The work presented in this article is partly sponsored by an Australian Research Council (ARC) grant. The work of Liyin Xue was also supported by an award from Macquarie University.

References

1. S. Adve and M. Hill. Weak ordering: a new definition. Proc. 17th Annual International Symposium on Computer Architecture, 1990, pp. 2–14.
2. P. Dewan. Architectures for collaborative applications. In M. Beaudouin-Lafon (ed.), Computer supported Co-operative Work, John Wiley & Sons, 1999, pp. 169–193.
3. P. Dourish. Consistency guarantees: exploiting application semantics for consistency management in a collaboration toolkit. In Proc. of the ACM Conference on CSCW, Nov. 1996, pp. 268–277.
4. C.A. Ellis and S.J. Gibbs. Concurrency control in groupware systems. In Proc. of ACM SIGMOD Conference on Management of Data, May 1989, pp. 399–407.
5. C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: some issues and experiences. In Communications of ACM 34(1), Jan.1991, pp. 39–58.
6. S. Greenberg and D. Marwood. Real time groupware as a distributed system: concurrency control and its effect on the interface. In Proc. ACM Conference on CSCW, November 1994, pp. 207–217.
7. G. E. Kaiser. Cooperative transactions for multi-user environments. In W. Kim (ed.), Modern Database Systems: the object model, interoperability, and beyond, ACM Press, 1994, pp. 409–433.
8. R. Kanawati. LICRA: A replicated-data management algorithm for distributed synchronous groupware application. In Parallel computing, vol. 22, 1997, pp. 1733–1746
9. Karsenty, C. Tronche, and M. Beaudouin-Lafon. GroupDesign: shared editing in a heterogeneous environment. In Usenix Journal of Computing Systems, 6(2), 1993, pp. 167–195.

10. T. P. Moran, K. McCall, B. van Melle, E. R. Pedersen, and F.G.H. Halasz. Some design principles for sharing in Tivoli, a white-board meeting support tool. In S. Greenberg, S. Hayne, and R. Rada (eds.), Groupware for Real-time Drawing: A Designer's guide, McGraw-Hill, 1995, pp. 24–36.
11. J. P. Munson and P. Dewan. A concurrency control framework for collaborative systems. In Proceedings of ACM Conference on CSCW, 1996, pp. 278–287.
12. C. R. Palmer and G. V. Cormack. Operation transforms for a distributed shared spreadsheet. In Proceedings of the ACM Conference on CSCW, 1998, pp. 69–78.
13. A. Prakash. Group editors. In M. Beaudouin-Lafon (ed.), Computer Supported Cooperative Work, John Wiley & Sons, 1999, pp. 103–133.
14. C. Sun and D. Chen. A multi-version approach to conflict resolution in distribute groupware systems. In Proceedings of International Conference on Distributed Computing Systems, April 2000.
15. C. Sun and C.A. Ellis. Operational transformation in real-time group editors: Issues, algorithms, and achievements. In Proceedings of ACM Conference on CSCW, Nov. 1998, pp. 59–68
16. C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. In ACM Transactions on Computer-Human Interaction, 5(1), March 1998, pp. 63–108.
17. L. Xue, K. Zhang, and C. Sun. Conflict control locking in distributed cooperative graphics editors. In Proceedings of the 1st International Conference on Web Information Systems Engineering (WISE 2000), Hong Kong, IEEE CS Press, June 2000, pp. 401–408.
18. L. Xue, K. Zhang, M. Orgun, and C. Sun. Version composition and identification schemes in distributed real-time groupware systems. In Macquarie Computing Reports, No. C/TR01-01, Macquarie University, February 2001
19. L. Xue, K. Zhang, and C. Sun. An integrated post-locking, multi-versioning, and transformation scheme for consistency maintenance in real-time group editors. In Proc. of the 5th Intentional Symposium on Autonomous Decentralised Systems, Dallas, Texas, USA, IEEE CS Press, Mar. 2001.
20. L. Xue, M. Orgun, and K. Zhang. A group-based time-stamping scheme for the preservation of group intentions. In Proceedings of the International Conference on DCW, Sydney, Australia, April 2002.

Multi Agent Transactional Negotiation: Application to E-marketing

V.K. Murthy and R. Abeydeera

School of Computer Science, UNSW@ADFA
University of New South Wales, Canberra ACT 2600 Australia
murthy@cs.adfa.edu.au

Abstract. We propose a multi-agent transactional paradigm based on object-based rule systems for realising distributed agent negotiation protocols in E-marketing. The construction of the protocol is carried out in two stages: first expressing a program into an object-based rule system and then converting the rule applications into a set of transactions on a database of active objects. Also, an algorithm to prove termination of the negotiation among the agents is described.

1 Introduction

This paper proposes a multi-agent transactional paradigm using object-based rule systems for realising distributed agent negotiation protocols in E-Commerce and E-marketing. The construction of the protocol is carried out in two stages: first expressing a program into an object-based rule system and then converting the rule applications into a set of transactions on a database of active objects. We also describe an algorithm to prove termination of the negotiation among the agents.

In a recent paper Fisher [6] describes a philosophical approach to the representation and execution of agent-based systems. This paper develops Fisher's philosophical approach into a concurrent multi-agent programming paradigm based on the transactional logic model [3]. The multi-agent system consists of the following subsystems, (Figure 1), Ishida [7], Dignum and Sierra [5].

1. *Worldly states or environment* U: Those states which completely describe the universe containing all the agents.
2. *Perception*: Depending upon the sensory capabilities (input interface to the universe or environment) an agent can partition U into a standard set of messages T, using a sensory function Perception ($PERCEPT$): $PERCEPT : U \rightarrow T$. $PERCEPT$ can involve various types of perception: see, read, hear, smell. The messages are assumed to be interpreted identically by all agents.
3. *Epistemic states or Mind* M: We assume that the agent has a mind M (that is essentially a problem domain knowledge consisting of an internal database for the problem domain data and a set of problem domain rules) that can be clearly understood by the agent without involving any sensory function. The database D sentences are in first order predicate calculus (also known

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 151–164, 2002.

as extensional database) and agents mental actions are viewed as inferences arising from the associated rules that result in an intentional database, that changes (revises or updates) D.

The agent's state of belief or a representation of an agent's state of belief, at a certain time, is represented by an ordered pair of elements (D, P). D is a set of beliefs about objects, their attributes and relationships stored as an internal database and P is a set of rules expressed as preconditions and consequences (conditions and actions). When T is input, if the conditions given in the left-hand side of P match T the elements from D that correspond to the right-side are taken from D and suitable actions are carried out locally (in M) as well as on the environment.

4. *Organizational Knowledge O:* Since each agent needs to communicate with the external world or other agents, we assume that O contains all the information about the relationships among the different agents. For example, the connectivity relationship for communication, the data dependencies between agents, interference among agents with respect to rules, information about the location of different domain rules are in O.
5. *INTRAN*: M is suitably revised or updated by the function called Internal transaction (*INTRAN*). Revision means acquisition of new information about the world state, while update means change of the agent's view of the world. Revision of M corresponds to a transformation of U, due to the occurrence of events, and transforming an agent's view due to acquisition of new information that modifies rules in P or their mode of application (deterministic, nondeterministic or probabilistic) and corresponding changes in database D. Updates to M correspond to changes in U due to the occurrence of events that changes D but not D or $INTRAN : M \times T \rightarrow M$.
6. *EXTRAN*: External action is defined through a function called global or external transaction (*EXTRAN*) that maps an epistemic state and a partition from an external state into an action performed by the agent. That is: $EXTRAN : M \times T \rightarrow A$; that is, current state of mind and new input activates an external action from A.
7. *EFFECT*: The agent also has an effectory capability on U by performing an action from a set of actions A (ask, tell, hear, read, write, speak, send, smell, taste, receive, silent), or more complex actions. Such actions are carried out according to a particular agent's role and governed by an etiquette, called protocols. The effect of these actions is defined by a function *EFFECT*, that modifies the world states through the actions of an agent: $EFFECT : A \times U \rightarrow U$; *EFFECT* can involve additions, deletions and modifications to U.

Thus an agent is a 9-tuple:

$$(U, T, M(P, D), O, A, PERCEPT, INTRAN, EXTRAN, EFFECT)$$

The interpreter repeatedly executes selected rules in P, until no rule can be fired.

We can interpret all the abstract machine models (Finite state machine or Turing machine) and parallel computational models (such as classifier systems)

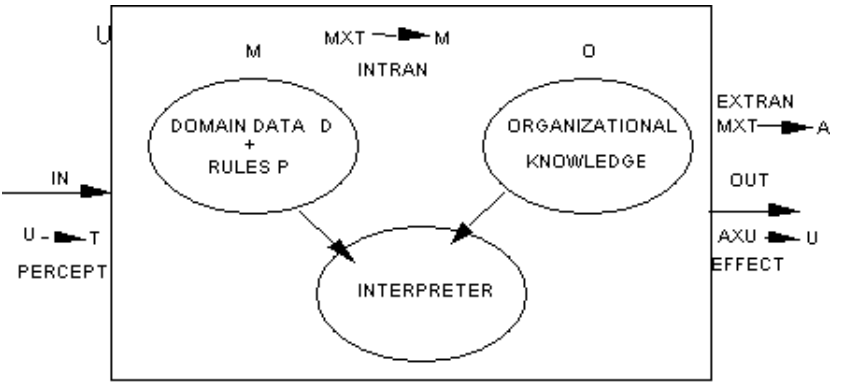


Fig. 1. A multi-agent system

as subclasses of the agents, by suitably formulating the definitions. Thus agents can exhibit the same power as a nondeterministic Turing machine.

The nature of internal production rules P, their mode of application and the action set A determines whether an agent is deterministic, nondeterministic, probabilistic or fuzzy. Rule application policy in a production system P can be modified by:

1. Assigning probabilities/fuzziness for applying the rule.
2. Assigning strength to each rule by using a measure of its past success.
3. Introducing a support for each rule by using a measure of its likely relevance to the current situation.

The above three factors provide for competition and cooperation among the different rules [9]. Such a model is useful in E-marketing for understanding how a new order, such as percolation (stock market crash or phase transition), can emerge from probabilistic and nondeterministic interactions between many agents.

2 What Is Negotiation?

"Negotiation" is an interactive process among a number of agents that results in varying degrees of cooperation, competition and ultimately to commitment that leads to a total agreement, consensus or a disagreement [5]. A negotiation protocol is viewed as a set of public rules that dictate the conduct of an agent with other agents to achieve a desired final outcome.

A negotiation protocol among agents involves the following actions or conversational states:

1. Propose: one puts forward for consideration a set of intentions called a proposal.
2. Accept: The proposal is accepted for execution into actions.
3. Refuse: The proposal is rejected for execution into actions.
4. Modify: This alters some of the intentions of the proposer and suggests a modified proposal- that is at the worst it can be a Refuse and a New proposal; or a partial acceptance and new additions.
5. No proposal: No negotiation.
6. Abort: Quit negotiation.
7. Report agreement: This is the termination point for negotiation in order to begin executing actions.
8. Report failure (agree to disagree): Negotiation breaks down.

Note that the above actions are not simple exchange of messages but may involve some intelligent or smart computation.

In BNF form, the negotiation has a syntax::

$$\begin{aligned} \textit{Negotiation} &:= \textit{Open Transaction} \mid \textit{Closed transaction} \\ \textit{Open Transaction} &:= \texttt{Propose} \mid \texttt{Accept} \mid \texttt{Refuse} \mid \texttt{Revise} \mid \texttt{No-proposal} \\ \textit{Closed Transaction} &:= \texttt{Abort} \mid \texttt{Agreement} \mid \texttt{Failure} \end{aligned}$$

A directed graph can be used to represent a negotiation process with the Open transaction as the initial state and the closed transaction as the final state. Such a directed graph that expresses the connectivity relationship among the agents can be real or conceptual and can be dynamic or static, depending upon the problem at hand.

Multi-agents can cooperate to achieve a common goal to complete a transaction to aid the customer. The negotiation follows rule-based strategies that are computed locally by its host server. Here competing offers are to be considered; occasionally cooperation may be required. Special rules may be needed to take care of risk factors, domain knowledge dependencies between attributes and positive and negative end conditions. During a transaction, several agents have to negotiate and converge to some final set of values that satisfies their common goal. Such a goal should also be cost effective so that it is in an agreed state at the minimum cost or a utility function.

To choose an optimal strategy each agent must build a plan of action and communicate with other agents. To illustrate this situation, we convert a distributed greedy algorithm that finds a minimal cost flight path in an airline route map (represented as a directed graph) into a negotiation problem (Section 6). In this case the system is deterministic and the negotiation reaches a stable state. However, it is possible that a system is nondeterministic or stochastic and has many stable states as in the E-market trading. In E-Market situation such a negotiation can lead to a speculation bubble or a crash, or a stagnation (Section 8) and a phase transition among such states.

3 Transactional Approach for Negotiation

Problem solving consists in finding a partially ordered sequence of application of desired operators that can transform a given initial state to a desired final state - called goal. Reaching a goal through a sequence of totally ordered sub-goals - is called a linear solution. For many problems a totally ordered concatenation of sub-goals may not exist. Such problems require a nondeterministic search. A solution based on a nondeterministic search is called a non-linear solution. Such a solution uses an act-verify strategy. Human problem solving also uses this strategy through preconditions and actions. When a human solves a problem, the solution process has a similarity to transaction handling problem; for each transaction is an exploratory non pre-programmed real-time procedure that uses a memory recall (Read), acquires a new information and performs a memory revision (Write). Each transaction is also in addition provided with the facility for repair (recovery-Undo) much like the repair process encountered in human problem solving. In human problem solving, several independent or dependent information is acquired from various knowledge sources and their consistency is verified before completing a step of the solution to achieve each sub-goal; this process corresponds to committing a sub-transaction in a distributed transaction processing system, before proceeding to reach the next level of sub-goal arranged in a hierarchy. Thus the transactional approach provides for a propose, act and verify strategy by offering a nonprocedural style of programming (called 'subjunctive programming') in which a hypothetical proposal or action (what if changes) is followed by verification, commitment or abort and restoration. So this paradigm is well suited for agent-based computations.

4 Object-Based Rules and Transactions

The multi-agent negotiation is based on Post production system model (and the related language) [13]. The production system paradigm occupies a prominent place in AI. This system consists of a set of rewrite rules consisting of a left-hand-side expression (LHS) and a right-hand side- expression (RHS). Given any string drawn from a database that matches the LHS, the corresponding RHS is substituted. Thus we may substitute expressions, constants or null elements permitting update, evaluation, insertion or deletion operations. The implementation of a production system operates in three-phase cycles: matching, selecting and execution. The cycle halts when the database fulfils a termination condition.

In order to use the production system as the basis for transactional multi-agent negotiation, we need to analyse how the rules (and hence the respective transactions) interfere with each other when they are applied. There are several ways in which the rules can interfere [7,8,9,10,11] when we consider object-based production rule systems (OPRUS). Here we assume that the rules do not interfere during the negotiation process or their interference is suitably controlled [7,8].

OPRUS can be implemented as transactions, by identifying every rule that fires as a transaction. We can find a direct correspondence between concurrent transaction processing systems [11] and concurrently enabled rule systems.

Therefore much of the known results in concurrency control in transaction processing can be directly applied.

5 Planning, Reasoning, and Negotiation

To solve a problem through negotiation, we need to look for a precondition that is a negation of the goal state and look for actions that can achieve the goal. This strategy is used widely in AI [13] and forms the basis to plan a negotiation. To systematically derive a multi-agent negotiation protocol (MAN) we use the specification approach [2]. Due to the lack of space we illustrate this approach through an Example in Section 6.1.

6 Design of an Agent Negotiation Protocol

A negotiation protocol should have the following properties:

1. The negotiation leads to a finite number of states.
2. Every kind of protocol message can be used.
3. The protocol leads to a negotiation process in which no message is unused.
4. The negotiation process does not loop and reaches a terminal state.

A protocol design has the following phases:

1. Identifying message types.
2. Explaining the possible sequences among the participants.
3. Identifying various conversational states.
4. Drawing the transition diagram.

A multi-agent system consists of a set of agents, a set of channels and a local memory for each agent. An agent can only read or write from its local memory. Channels are assumed error free and deliver messages in the order they were sent. For each channel there is exactly one agent that sends messages along that channel and exactly one agent that receives the messages across that channel. Associated with each channel is a buffer. For each channel the only action that can be taken by the sending agent is to send a message (data message and other messages) if the buffer is not full, and the only action that can be taken by the receiving agent is to receive the message, if the buffer is not empty.

We now describe how to carry out distributed multi-agent negotiation by sending, receiving, handshaking and acknowledging messages and performing some local computations. A multi-agent negotiation has the following features [4, 5,6,14,15,16,17,18]:

1. There is a seeding agent who initiates the negotiation.
2. Each agent can be active or inactive.
3. Initially all agents are inactive except for a specified seeding agent that initiates the computation.

4. An active agent can do local computation, send and receive messages and can spontaneously become inactive.
5. An inactive agent becomes active if and only if it receives a message.
6. Each agent may retain its current belief or revise its belief as a result of receiving a new message by performing a local computation. If it revises its belief, it communicates its revised state of belief to other concerned agents; else it does not revise its solution and remains silent.

To illustrate the design of a negotiation protocol we consider the lowest cost path problem in a graph.

6.1 Example

Consider the problem of finding a lowest cost path between any two vertices in a directed graph whose edges have a certain assigned positive costs (Figure 2). The lowest cost path problem requires the entity set of vertices, the relationship set of ordered pairs of vertices (x, y) representing edges, and the attribute of cost c for each member of the relationship set, denoted by (x, y, c). Given a graph G the program should give for each pair of vertices (x, y) the smallest sum of costs path from x to y. The vertex from which the lowest cost paths to other vertices are required is called the root vertex r (vertex 1 in this example). Let s denote the sum of costs along the path from the root to y; we assume that c (and hence s) is positive. This information is described by the ordered 4-tuple (x, y, c, s): (*vertex label*, *vertex label*, *cost*, *sum of costs from root*). The fourth member of the 4-tuple, namely the sum of costs from a specified root remains initially undefined and we set this to a large number $*$. We then use the production rules to modify these tuples or to remove them.

To find the lowest cost path to all vertices from a specified root r, we use the MAN for tuple processing and let the 4-tuples interact; this interaction results in either the generation of modified 4-tuples or the removal of some 4-tuples of the representation.

Specification to find the shortest path

Let $C(i, j)$ be the cost of path (i, j). A better path is one that can pass through some vertex k such that:

$$C(i, k) + C(k, j) < C(i, j)$$

That is our production rule is: If $C(i, k) + C(k, j) < C(i, j)$ then delete $C(i, j)$ and set $C(i, j) = C(i, k) + C(k, j)$. The invariant is: if $C(i, j)$ is the initial cost then all the costs are always less than or equal to $C(i, j)$. We refine this by using the rule:

If $C(i, k) < C(p, k)$, delete $C(p, k)$ and retain $C(i, k)$. Thus the following three production rules result:

Rule 1: If there are tuples of the form $(r, r, 0, 0)$ and $(r, y, c, *)$, replace $(r, y, c, *)$ by (r, y, c, c) and retain $(r, r, 0, 0)$.

Rule 1 defines the sum of costs for vertices adjacent to the root, by deleting $*$ and defining the values.

Rule 2: If there are tuples of the form (x, y, c_1, s_1) and (y, z, c_2, s_2), where $s2 > s1 + c2$ then replace (y, z, c_2, s_2) by $(y, z, c_2, s_1 + c_2)$; else do nothing.
Rule 2 states that if $s_2 > s_1 + c_2$ we can find a lower cost path to z through y.

Rule 3: If there are tuples of the form (x, y, c_1, s_1) and (z, y, c_2, s_2) and if $s_1 < s_2$, then remove (z, y, c_2, s_2) from the tuple set; else do nothing.
Rule 3 states that for a given vertex y which has two paths, one from x and another from z, we can eliminate that 4-tuple that has a higher sum of costs from the root. The above three rules provide for nondeterministic local computation by many agents and we are left with those tuples that describe precisely the lowest cost path from the root. Note that here *INTRAN* only updates D; but does not revise P.

We assume that there are n agents with names identical to the nodes in the graph and each agent is connected to other agents in an isomorphic manner to the given graph. Such an assumption on the topology of the network simplifies the organizational knowledge O. Using O, each agent knows the identity of its neighbours, the direction and cost of connection of the outgoing edges. Thus for the given directed graph the outdegree of each node is the number of sending channels and the indegree is the number of receiving channels.

The revised production rules for multi-agent computation are as follows:

Initialisation of beliefs: Agent 1 (root) sends to all its neighbours x the tuple $(1, x, c, c)$ describing the name of the root, and the distance of x from the root c; all the neighbours of the root handshake, receive, and store it.

Initial set of beliefs: Each agent x sends its neighbour y at a distance c_1 from it, the tuple $(x, y, c_1, c + c_1)$ describing its name, its distance to y and the distance of y from the root through x using its distance to the root c.

Update of beliefs: Each agent y compares an earlier tuple (x, y, c_1, s_1) got from a neighbour x, or the root, with the new tuple (z, y, c_1', s_1') from another neighbour z. If $s_1 < s_1'$, then y retains (x, y, c_1, s_1) and remains silent; else it stores (z, y, c_1', s_1') and sends out the tuple $(y, w, c_2, s_1' + c_2)$ to its neighbour w at a distance c_2, advising w to revise its distance from the root.

Stability and halting: An agent does not send messages if it receives a message from another agent that tells a higher value for its distance from the root and ignores the message, i.e., it does not revise its beliefs. Thus it contains only the lowest distance from the root. All the agents halt when no more messages are in circulation and the system stabilizes.(For termination of negotiation see next section).

Consider the directed graph in Figure 2, in which the edge costs are as shown; we denote the graph by the triplet, a pair of nodes (x, y) followed by the cost c of the edge, hence (x, y, c). The graph in Figure 2 is then given by:

$$(1,2,50)\ (1,3,10)\ (1,5,45)\ (2,3,15)\ (2,5,10)$$
$$(3,4,15)\ (4,2,20)\ (4,5,35)\ (5,4,30)\ (6,4,3)$$

We choose the vertex 1 as the root; we use the following format for representing the distances in the graph: (vertex label, vertex label, cost, sum of costs from root).

Thus the graph is encoded in the form:

$$(1,1,0,0)\quad (1,2,50,*)\ (1,3,10,*)\ (1,5,45,*)\ (2,3,15,*)$$
$$(2,5,10,*)\ (3,4,15,*)\ (4,2,20,*)\ (4,5,35,*)\ (5,4,30,*)$$
$$(6,4,3,*)$$

We then apply the three rules systematically. This results in the following tuples that describe the lowest cost path subgraph:

$$(1,1,0,0)\ (1,3,10,10)\ (1,5,45,45)\ (3,4,15,25)\ (4,2,20,45)$$

Note that the 4-tuple $(6,4,3,*)$ gets eliminated as vertex 6 cannot be reached from the root vertex 1. Figure 3 illustrates the computation as well as communication in this process, along with the negotiation termination algorithm to be described in Section 7.

7 Agent Negotiation Termination Detection

We now describe an algorithm, called "Commission-Savings-Tally Algorithm (COSTA)", for the global termination detection of a negotiation protocol. This is a general algorithm. We will illustrate the use of this algorithm to find the shortest path in a graph of Figure 2, described in Section 6, using agent negotiation. The agent negotiation algorithm terminates or stabilizes with appropriate distances as shown in Figure 3.

Let us assume that the N agents are connected through a communication network represented by a directed graph G with N nodes and M directed arcs, as in Figure 2. Let us also denote the outdegree of each node i by $Oud(i)$ and indegree by $Ind(i)$. Also we assume that an initiator or a seeding agent exists to initiate the transactions. The seeding agent (SA) holds an initial amount of money C. When the SA sends a data message to other agents, it pays a commission:

$C/(Oud(SA)+1)$ to each of its agents and retains the same amount for itself. When an agent receives a credit it does the following:

1. Let agent j receive a credit $C(M(i))$ due to some data message $M(i)$ sent from agent i. If j passes on data messages to other agents j retains for its credit $C((M(i))/(Oud(j)+1)$ and distributes the remaining amount to other $Oud(j)$ agents. If there is no data message from agent j to others, then j credits $C(M(i))$ for that message in its own savings account; but this savings will not be passed on to any other agent, even if some other message is received eventually from another agent.
2. When no messages are received and no messages are sent out by every agent, it waits for a time-out and sends or broadcasts or writes on a transactional blackboard its savings account balance to the initiator.
3. The initiator on receiving the message broadcast adds up all the agents' savings account and its own and verifies whether the total tallies to C.

4. In order to store savings and transmit commission we use an ordered pair of integers to denote a rational number and assume that each agent has a provision to handle exact rational arithmetic. For convenience, if we choose $C = 1$, it is sufficient to store the denominator of the rational number remembering that the actual value is a reciprocal for summing over the credits.

We prove the following theorems to describe the validity of the above algorithm:

Theorem 1. *If there are cycles present among the agents (including the initiator itself) then the initiator cannot tally its sum to C within a finite time period. Hence negotiation fails and the algorithm has to abort after a properly chosen time-out period.*

Proof. Assume that there are two agents i and j engaged in a rule dependent argument cycle. This means i and j are revising their beliefs forever without coming to an agreement. Let the initial credit of i be x. If i passes a message to j, then i holds $x/2$ and j gets $x/2$. If eventually j passes a message to i, then its credit is $x/4$ and i has a credit $3x/4$; if there is continuous exchange of messages for ever then their total credit remains $(x - x/2k)$ with $x/2k$ being carried away by the message at the k-th exchange. Hence the total sum will never tally in a finite time period.

Theorem 2. *The negotiation terminates within a finite time-out period, if and only if, the initiator tallies the sum of all the agents savings to C. In other words, there is no situation in which the sum has tallied to C but negotiation is incomplete, or sum has not tallied to C but the negotiation is completed.*

Proof. If part: If the initiator tallies the sum to C within a finite time out period, it implies that all the agents have sent their savings and no message is in transit carrying some credit and there is no chattering among agents.

Only if part: The credit assigned can be only distributed in the following manner:

1. An agent has received a message and credit in a buffer; if it has sent a message then a part of the credit is lost; else it holds the credit in savings.
2. Each message carries a credit; so, if a message is lost in transit or communication fails then total credit cannot be recovered.

Thus negotiation terminates within a properly chosen time-out period when the total sum tallies to C; then all the agents have reached an agreement on their beliefs.

7.1 Example

The combined communication protocol and computational tree of Figure 3 is obtained from Figure 2, using the rules 1, 2, 3 of Example 6.1. At initiation, the node labelled 1 is the root and the seeding agent. It contains the pair $(0, 0)$

indicating that it is the root and its distance to the root through itself is zero. It transmits the information to each neighbour its distance from the neighbour and the distance of its neighbour to the root through itself. Also it starts with a credit 1and retains a credit of $1/(Oud(SA)+1$ to itself, and transmits the same amount to its neighbours 2, 3, 5 which in this case is 1/4. Along each edge from each node x to node y the credits transmitted are indicated. The retained credit for each transmission is indicated near the node. Then the COSTA proceeds as indicated generating the communication tree of Figure 3. Note that in this process, agent node 2 revises its earlier belief from the new message received from 4; but the other nodes 3, 4, 5 do not revise their initial beliefs and remain silent, since the later message received by them contained a longer distance path than what they received earlier from node 1. Finally as indicated in the first four rules in this section we sum over all the retained credits after each transmission. These are respectively:

Node 1: 18/72; Node 2: 7/72; Node 3: 16/72; Node 4: 12/72; Node 5: 19/72. Note that the sum tallies to one. Also the shortest distance to the root from each node is:

Node 1: 0; Node 2: 45; Node 3: 10; Node 4: 25; Node 5: 45.

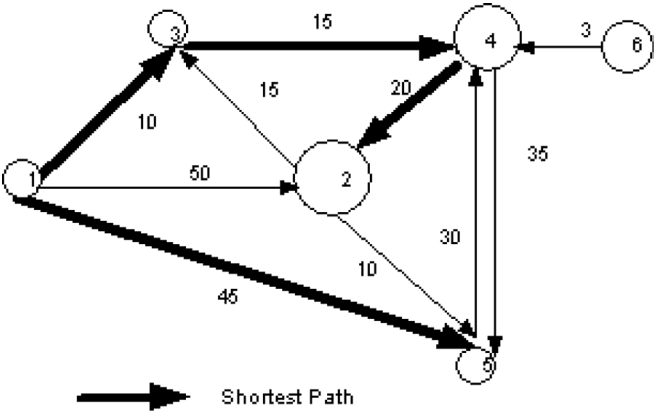


Fig. 2. Shortest path

8 Application of Agents to E-market Modelling

We now describe how to model an E-market with many traders, called buyers and sellers, using agents. These agents negotiate over the internet to sell or buy shares or stocks in a stock market. In E-market situation, it is possible that the negotiation ultimately leads to self organization and criticality causing crashes. That is individual agents which correspond to a microscopic system can

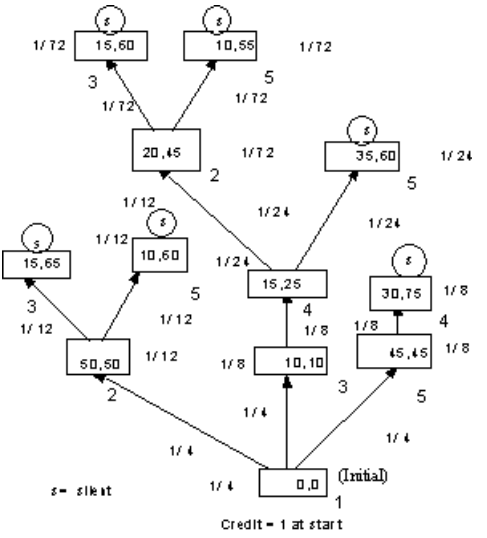


Fig. 3. Multi-agent negotiation

emerge as a self organizing macroscopic system corresponding to a "percolation model" [1,12].

In an E-market situation (see Figure 1), to start with, the domain data D, rules P and organizational knowledge O can be based on three factors:

1. The experience and economics knowledge of an agent deployed by a trader based totally on individualistic idiosyncratic criteria;
2. The trader's acquired knowledge through communication with other selected agents; such a trader is called a fundamentalist.
3. The trader's acquired knowledge by observing the trends on market from a collective opinion of other traders; such a trader is called a trend chaser.

In practice a trader is influenced by all the above factors and the modified knowledge is incorporated in D, P and O.

In E-market at every time instant a trader can adopt three possible states of action: Buy, Sell or Wait respectively represented by three states 1, -1, 0. Each agent corresponding to a trader can communicate with one another and this creates a conceptual bond or connectivity relationship among them modifying the organizational knowledge O. This bond is created with a certain probability determined by single parameter which characterises the willingness of an agent to comply with others. Since detailed information about the mechanism of bond formation is difficult to know, we can assume that any two agents are randomly connected with a certain probability. This divides the agents into clusters of

different sizes whose members are linked either directly or indirectly via a chain of intermediate agents. These groups are coalitions of market participants who share the same opinion about their activity. The decision of each group is independent of its size and the decision taken by other clusters.

Using percolation theory, it can be shown that when every trader is on average connected to another, more and more traders join the spanning cluster, and the cluster begins to dominate the overall behaviour of the system. This gives rise to a "speculation bubble" (if the members all decide to buy) and a crash (if the members all decide to sell) or a stagnation (if the members all decide to wait). Crash is a highly cooperative phenomenon and depends upon trading rules, the speed and volume of the exchange of information, and the connectivity relationship. Thus, an analogy exists between stock-market crashes and critical phenomena or phase transitions in physics [1,12].

9 Concluding Remarks

The multi-agent negotiation paradigm has the following desirable features:

1. It provides a programming methodology free from control management. The transactional implementation of rules provides for propose-verify-revise strategy.
2. It describes the application of locality principle in protocol construction.
3. It will be applicable to programming as well as design of multi-agent architectures consisting of agents that serve as processes, functions, relations or constraints depending upon the context.

We have implemented the above multi-agent negotiation paradigm using the Java-based agentTool toolkit, which is a graphical development environment [4].

References

1. Bak, P: How Nature works: The Science of Self-organized criticality, Springer-Verlag, New York (1996).
2. Bauer, F.L and Brauer, W: Logic and Algebra of Specification, Springer-Verlag, New York (1993)
3. Bonner, A.J., Kifer, M: Application of transaction logic to knowledge representation, Lecture Notes in Computer Science, Temporal Logic, Vol.827, Springer-Verlag, New York (1994) 67–81
4. DeLoach, S.A., Wood., M., Sparkman, C: Multiagent Systems Engineering, International Journal of Software Engineering and Knowledge Engineering, 11, 2001, 231–258
5. Dignum, F., Sierra, C: Agent Mediated E-Commerce, Lecture notes in Artificial Intelligence, Vol. 1991; Vol. 2003, Springer Verlag, New York (2001)
6. Fisher, M: Representing and executing agent-based systems, in Intelligent Agents, Woolridge, M., and Jennings, N.R (Eds.), Lecture Notes in Computer Science, Vol. 890, Springer-Verlag, New York(1995) 307–323

7. Ishida, T: Parallel, Distributed and Multiagent Production Systems, Lecture Notes in Computer Science, Vol. 878, Springer Verlag, New York (1994)
8. Kuo, S., Moldovan, D: The state of the art in parallel production systems, J. Parallel and distributed computing, 15 (1992) 1–26
9. Murthy, V.K., Krishnamurthy, E.V: Probabilistic Parallel Programming based on multiset transformation, Future Generation Computer Systems, 11(1995) 283–293
10. Ozsu, M.T: Transaction Models and Transaction management in Object-oriented Database Management Systems, in Advances in Object-oriented Database Systems, Computer and System Sciences, Vol.30, Springer-Verlag, New York (1994)
11. Paton, N.W., and Williams, M.H: Rules in database Systems, Springer-Verlag, New York (1994)
12. Paul, W., Baschnagel, J: Stochastic Processes, Springer-Verlag, New York(2000)
13. Rich, E., Knight, K: Artificial Intelligence, McGraw Hill, New York (1991)
14. Singh, M.P: Multiagent Systems, Lecture Notes in Computer Science, Vol. 799, Springer-Verlag, New York (1994)
15. Van de Velde, W., Perram, J.W: (Eds.), Agents Breaking Away, Lecture Notes in Computer Science, Vol. 1038, Springer-Verlag, New York (1996)
16. Wittig, T: ARCHON: An architecture for multiagent systems, Ellis Horwood, New York (1992)
17. Woolridge, M., Jennings, N.R: Agent theories, Architectures and Languages, A survey, Lecture Notes in Computer Science, Vol. 890, Springer-Verlag, New York, (1995), 1–39
18. Wooldridge, M., Muller, J.P., Tambe, M: (Eds.), Intelligent Agents II, Lecture Notes in Computer Science, Vol. 1037, Springer-Verlag, New York (1996)

A Rule-Driven Approach for Defining the Behaviour of Negotiating Software Agents*

Morad Benyoucef[1], Hakim Alj[1], Kim Levy[1], and Rudolf K. Keller[2]

[1] Département d'informatique et de recherche opérationnelle
Université de Montréal, C.P. 6128 Succursale Centre-ville
Montréal, Québec, H3C 3J7, Canada
{benyouce, aljh, levyk}@iro.umontreal.ca
[2] Zuehlke Engineering AG, Wiesenstrasse 10a
CH-8952 Schlieren, Switzerland
ruk@zuehlke.com

Abstract. One problem with existing agent-mediated negotiation systems is that they rely on ad hoc, static, non-adaptive, and hardcoded schemes to represent the behavior of agents. This limitation is probably due to the complexity of the negotiation task itself. Indeed, while negotiating, software (human) agents face tough decisions. These decisions are based not only on the information made available by the negotiation server, but on the behavior of the other participants in the negotiation process as well. The information and the behavior in question are constantly changing and highly uncertain. In this paper, we propose a rule-driven approach to represent, manage and explore negotiation strategies and coordination information. Among the many advantages of this solution, we can cite the high level of abstraction, the closeness to human understanding, the versatility, and the possibility to modify the agents' behavior during the negotiation. To validate our approach, we ran several agent tournaments, and used a rule-driven mechanism to implement bidding strategies that are common in the English and Dutch auctions. We also implemented simple coordination schemes across several auctions. The ongoing validation work is detailed and discussed in the second part of the paper.

1 Introduction

According to Kephart et al., over the course of the next decade, the global economy and the Internet will merge into an information economy bustling with billions of autonomous software agents that exchange information goods and services with humans and other agents [1]. In addition to exchanging information, software agents can be programmed to search, compare, learn, negotiate,

* The completion of this research was made possible thanks to Bell Canada's support through its Bell University Laboratories R&D program, funding by the NSERC (National Sciences and Engineering Research Council of Canada, CRD-224950-99), and support by the CIRANO (Centre Interuniversitaire de Recherche en ANalyse des Organisations).

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 165–181, 2002.

and collaborate [2], making them particularly useful for the information-rich and process-rich environment of electronic commerce (e-commerce) [3]. As e-commerce usually involves information filtering and retrieval, personalized evaluation, complex coordination, and time-based interaction (among other things), it can greatly benefit from the introduction of software agents. Therefore, we talk of agent-mediated e-commerce. A simple e-commerce transaction can be seen as a three-phase scenario: (1) finding a partner for the transaction; (2) negotiating the terms of the transaction using a recursive process; (3) and carrying out the transaction. We are interested in the negotiation phase, and particularly in its automation by way of software agents capable of mimicking some of the behavior of human negotiators. We believe that agent-mediated negotiation absorbs many of the costs and inconveniences of manual negotiation [4].

The negotiation process itself is a form of interaction made of *protocols* and *strategies*. The protocols comprise the rules (i.e., the valid actions) of the game, and, for a given protocol, a participant (human or software) uses a strategy (i.e., a plan of action) to maximize her utility [5]. Based on this, many strategy-enabled agent-mediated negotiation systems have been described in the literature. Unfortunately, most of them use hardcoded, predefined, and non-adaptive negotiation strategies, which is evidently insufficient in regard to the ambitions and growing importance of automated negotiations research. The well-known KASBAH agent marketplace [6] is a good example of such systems. To overcome this shortcoming, we believe that negotiation strategies should be treated as declarative knowledge, and could, for instance, be represented as *if-then* rules, and exploited using inference engines.

The focus of our research is on *combined negotiations* [7], a case where the consumer combines negotiations for different complementary items that are not negotiated on the same server. For instance, a consumer may want to simultaneously purchase an item and its delivery by engaging in separate negotiations. If software agents are assigned to these negotiations, this poses a coordination problem between them. Many multi-agent negotiation systems found in the literature still rely on ad hoc schemes to solve this problem [8,9]. Again, we believe that a declarative approach can be used to describe and manage agent coordination across several negotiations.

To validate our approach, we designed and implemented an automated negotiation system called CONSENSUS [7] that enables a human user to instantiate one or more software agents, provide them with negotiation strategies and coordination know-how, register them on corresponding negotiation servers, and launch them. The agents use the strategies to negotiate according to the protocol dictated by the server, and the coordination know-how to coordinate their actions. An example of a strategy, applicable to an English auction (one of many existing negotiation protocols) is: "if you notice any form of jump bidding, then quit". Jump bidding means making a bid that is far greater than necessary in order to signal one's interest in the auctioned item (see Section 4). An example of coordination know-how, applicable to two agents bidding as partners in two separate auctions for two complementary items is: "if your partner loses in its

auction, then stop bidding and wait for further instructions" (see Section 4). We are currently testing various strategies and coordination schemes by way of agent tournaments. A large part of the paper is dedicated to this ongoing work.

Section 2 of the paper gives some background on strategy-enabled agent-mediated e-negotiations, and then presents our approach. Section 3 details our view of agent coordination after reviewing some related work. Section 4 is dedicated to the validation tests we are currently conducting. We wrap up the paper with a conclusion in Section 5.

2 Negotiation Strategies

2.1 Agent-Mediated E-negotiation

Electronic negotiation (e-negotiation) takes place when the negotiating function is performed by (networked) computers. Fully automated e-negotiation requires that all parties involved be software agents, semi-automated e-negotiation involves a human negotiating with a software agent, and manual e-negotiation refers to processes in which all parties are human [10]. Within fully automated e-negotiation, Parkes et al. identify autonomous and semi-autonomous agents. The former require complete preferences in order to represent the user, the latter only bid when they have enough knowledge, and query the user when their best action is ill-defined given the current information [4]. Agents can be involved in competitive negotiations when they have conflicting interests. They can be involved in cooperative negotiations when they all aim at satisfying the same interest [11].

In addition to their use in e-negotiations, agents are actually used in other phases of e-commerce transactions, including shopping, advertising, delivery, marketing and sales analysis [1]. Pricebots, for instance, are software agents that employ price-setting algorithms in an attempt to maximize profits, thus helping sellers to increase flexibility in their pricing strategies. An example taken from [12] points to `books.com`, which implements real-time dynamic pricing by using pricebots to monitor prices on competitor sites and offer the customer a lower price. Shopbots, to cite another example, are agents that gather information from multiple on-line vendors about the price and quality of goods and services [12]. One such system is `mysimon.com`.

2.2 Challenges of Strategy-Enabled Negotiation Systems

Designing, building, and tuning software agents before letting them lose in widely competitive scenarios like e-negotiations, inhabited by (human and software) expert negotiators, happens to be an arduous task [13]. This is why most strategy-enabled agent-based systems use predefined and non-adaptive negotiation strategies in the generation of offers and counteroffers [14]. Some commercial online auction sites such as eBay.com, offer the possibility of *proxy bidding* — i.e., an agent with a simple strategy: "bid until you reach your reserve price, by going

up each time with a certain increment." On the academic front, the buying (selling) agents of the KASBAH marketplace can choose between three negotiation strategies: anxious, cool-headed and frugal, corresponding to linear, quadratic, and exponential functions, respectively, for increasing (or decreasing) their bid (or ask price) over time [6].

Negotiating agents face tough decisions such as whether or not to accept a bid, whether or not to bid, how much to bid, etc. Those decisions must profit from all the information available in the marketplace: description of the good, its expected resale value, price history, participants' identities and behavior, etc. [13] This information is constantly changing and highly uncertain — new goods become available, buyers come and leave, prices change, etc. To complicate things further, the participants' goals, beliefs, intentions are expected to change over time [15].

A successful strategy must take into account the strategies of the opponents [13] as well as their reputation [1]. It must also protect against the opponents trying to extract the agent's private information. In a bargaining situation for instance, the buyer agent usually knows its owner's willingness-to-pay for an item [16]. If the seller agent (human or software) discovers this information, it can make a take-it-or-leave-it offer that will extract the buyer's entire surplus. Finally, we should mention that, with eBay.com's proxy bidding, one must reveal the highest price one is willing to pay, thus giving the site information that could be used to cheat the bidder [9].

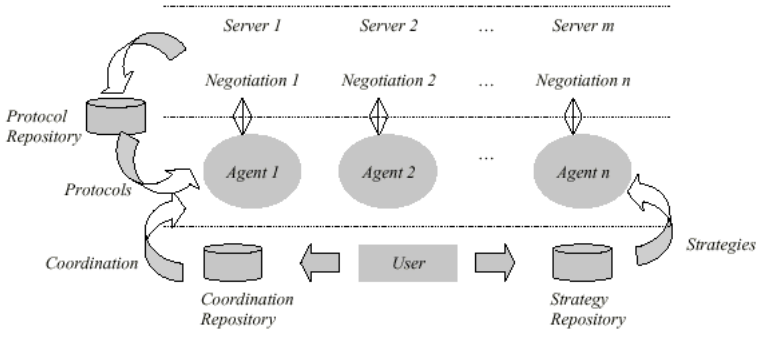


Fig. 1. Protocols, Strategies and Coordination in CONSENSUS

2.3 The CONSENSUS Approach

As mentioned in Section 1, the basic components of automated negotiation systems are the protocol and strategies [17]. The protocol defines the interaction between the agents. The strategies specify the sequence of actions the agent plans to make during the negotiation. In CONSENSUS, the protocol (i.e., the

negotiation rules) is communicated to the participants before they engage in the negotiation. A human negotiator would consult the protocol; a software agent would download it in a usable format (XML for instance). However, the strategies are the responsibility of the negotiator. A human negotiator would use her strategies to make offers or to respond to her opponents' offers; a software agent would do the same, based on the instructions provided by its creator. In addition to protocols and strategies, we introduce a third component: coordination. This is the information that the agents need in order to coordinate their actions while they participate in a combined negotiation (see Section 1). Figure 1 presents our view of the three components.

Two approaches inspired our representation for the strategies. Su et al. use a high-level rule specification language and GUI tools to allow human negotiation experts to add and change negotiation strategies at run-time [18]. Each strategy is expressed as an Event-Trigger Rule, which specifies the condition to be checked and the actions to be taken by the negotiation server. Rules are activated upon the occurrence of specific events during the negotiation process. Although not directly related to e-negotiations, the work of Grosof et al. on contract representation takes a similar approach [19]: since contract terms usually involve conditional relationships, they are expressed as rules. Similarly, we see strategies as rules that can be fed to the agents before and/or during the negotiation. Strategies such as "if the bidding gets too intense, then abandon the negotiation", and "if you notice any jump-bidding, shielding, or sniping, then wait for further instructions" can indeed be nicely coded as rules (see Section 4).

Rules drive the activity in the agents by describing the action(s) to take when specified conditions are met [21]. They are at a relatively high level of abstraction, and are closer to human understanding, especially by domain experts who are typically non-programmers. They are relatively easy to modify dynamically [22], and new behavior can be specified by adding rules, without modifying the previous ones. Furthermore, rules can be augmented with procedural attachments so that they have an effect beyond pure-belief inferring [23]. Finally, off-the-shelf rule engines can be used to implement rule-based systems. Figure 2 shows two rules in the ILOG JRules syntax [24], which implement proxy bidding. Rule1 is triggered when the agent is trailing while its reserve price is not met. In this case it places a bid equal to the actual bid plus the increment. Rule2 enables it to quit whenever its reserve price is met.

3 Coordination of Negotiating Agents

3.1 Background and Related Work

The interaction between agents can vary from simple information interchanges, to requests for particular actions to be performed and on to cooperation (working together to achieve a common objective) and coordination (arranging for related activities to be performed in a coherent manner) [25]. Multiple agents need to be coordinated to prevent anarchy; meet global constraints; distribute expertise, resources or information; deal with dependencies between agents; and ensure

```
// If you are not leading and you have not
// reached your reserve price then bid.
Rule Rule1
{
  When { ?x: TheElement(iLead == false;
         (highestBid+myIncrement) <= myReservePrice);}
  Then { modify ?x {action = "BID";} }
};
// If you are not leading and your bid is greater
// than the reserve price, then quit.
Rule Rule2
{
  When { ?x: TheElement(iLead == false;
         (highestBid + myIncrement) > myReservePrice);}
  Then { modify ?x {action = "DROP";} }
};
```

Fig. 2. Proxy bidding in the JRules Syntax

efficiency [15]. In CONSENSUS, there is clearly a need to coordinate the software agents (see Section 1). First, in a situation where many agents participate in separate negotiations with the goal of purchasing one item (e.g., they engage in many concert ticket auctions with the goal of purchasing one ticket), we need to make sure that only one agent finalizes its deal, and that only the agent in the cheapest auction (i.e., the one with the smallest asking price) is the one to bid. It is common in online auctions to forbid bidders from breaking their commitments. Thus, if more than one agent end up winning their auctions, they would not be able to retract, or at best they would be, but they would pay a penalty. Second, in a situation where multiple agents negotiate a package of complementary items (i.e., there is no use for one item without the others), we need to make sure that all the items or none are purchased in order to avoid *exposure*. Exposure occurs when we need many items, enter negotiations for all items, and end up making a deal on some but failing to make a deal on others. Coordination in CONSENSUS will be discussed further, but first, we review some related work.

The Biddingbot [8] is a Multi-Agent System (MAS) that supports users in monitoring multiple auctions. It consists of one leader agent and several bidder agents, each one being assigned to an auction. Bidder agents gather information, monitor, and bid in the multiple sites simultaneously. The leader agent manages cooperation among them, sends user requests to them, and presents bidding information to the user.

Another system, designed at the HP Labs in Bristol, UK, takes a different approach. Instead of a MAS, it uses one single agent [9]. The agent participates in many auctions for one item, and coordinates bids across them using a two-part algorithm: (1) a coordination component, which ensures it has the lowest leading bids; and (2) a belief-based learning and utility analysis component to

determine if it should deliberately lose an auction in the hope of doing better in another one later.

Not directly related to e-negotiation research, but highly relevant in the way it deals with coordination in MAS is the ARCHON project [26]. Software agents are divided into two layers: an ARCHON layer and an application program. The former encapsulates knowledge about cooperation, which is domain independent and encoded in terms of production rules. The latter performs the problem solving process. The separation of cooperation knowledge from application knowledge enables generic cooperation features to be reused for other applications.

3.2 The CONSENSUS Approach

Inspired by the ARCHON approach, we treat coordination information as declarative knowledge, and represent it as if-then rules which the agents exploit using an inference engine. Since our agents are already coupled with rule engines (for their strategy component), it is convenient to use the same rule engine to execute the coordination. We distinguish between strategy rules (described in Section 2), used to determine the action to take based on the information about a particular negotiation, and coordination rules, which are used to determine the action to take based on information about other negotiations (possibly combined with information about the negotiation itself). Coordination rules, as well as strategy rules have conditions that are built on the state of the agent, the amount (spent, to spend, committed, remaining, etc.), the remaining time in the negotiation, the frequency of bids, etc.

Suppose we have two agents participating in separate negotiations with the goal of purchasing just one item. The following rule ensures that Agent1 does not commit itself if Agent2 is already committed: "if Agent2 is leading or is in the process of bidding, then Agent1 should wait." Figure 3 shows the rule in the JRules syntax. Note that this rule is too restrictive since an agent cannot make a bid if the other one is leading. Evidently, this should not always be the case. If breaking commitments is allowed (perhaps at a certain cost) the rule in question can be relaxed, and we may have the two agents leading at the same time while making sure that one of them drops out of the auction before it is too late.

Suppose now that we have two agents participating in separate negotiations with the goal of purchasing two complementary items. The following rule minimizes the risk of exposure (see Section 3.1): "if Agent2 is trailing, and its chances of making a deal are slim, then Agent1 should wait for further instructions."

Finally, for practical reasons, we adopted a coordination approach where the agents post and read from a general *blackboard*. Coordination rules are therefore triggered by information that the agents make available in the blackboard (see Figure 3). This approach is suitable only if the tasks are assigned, a priori, to the agents, if the number of agents is small, and if the agents share a common domain understanding. Evidently, all requirements are satisfied in our case, otherwise, the blackboard scheme would result in a severe bottleneck.

```
// If Agent2 is leading, or is in the process of bidding,
// then Agent1 waits
Rule coordinate1
{
  Priority = high;
  When { ?y: TheElement(iLead == false);
         Blackboard(?b1: get("u2","iLead");
                    ?b2: get("u2","iBid"))
         from getBlackboard(); evaluate(?b1 || ?b2); }
  Then { modify ?y
         { action = "DO NOTHING";}
         assert logical BidBloc() {ref = new Integer (0);} }
};
```

Fig. 3. A coordination rule in the JRules syntax

4 Validation

In order to validate our choice of representation for strategies and coordination, we used our agent-mediated negotiation system (i.e., CONSENSUS) to conduct bidding tournaments. The agents function in repeated cycles we call pulses. A pulse is made of four steps as described by the following piece of code:

```
Repeat
  Sleep(p);
  GetInformation();
  Think();
  Act();
Until (State = winning or State = loosing or State = dropping)
```

The pulse's steps are: (1) `sleep(p)`: the agent goes to sleep for a period p that can be fixed or determined dynamically; (2) `getInformation()`: the agent queries the server and updates its private information; (3) `think()`: the agent applies the rules to determine the action to take; and (4) `act()`: the agent takes the appropriate action. The possible actions are: (1) `do-nothing`: take no action for the time being; (2) `bid(v)`: bid the amount v; and (3) `drop`: quit permanently. Finally, the agent's states are: `trailing`, `bidding`, `leading`, `winning`, `losing`, and `dropping`.

Each agent instantiates a rule engine, enabling it to exploit the rules. There are three categories of rules: (1) basic rules, which determine the agent's behaviour within the negotiation protocol at hand; (2) strategy rules; and (3) coordination rules. Using this setting, we conducted bidding tournaments within a simulated market, and we present here the results involving the English and Dutch auctions, and some coordination schemes across them.

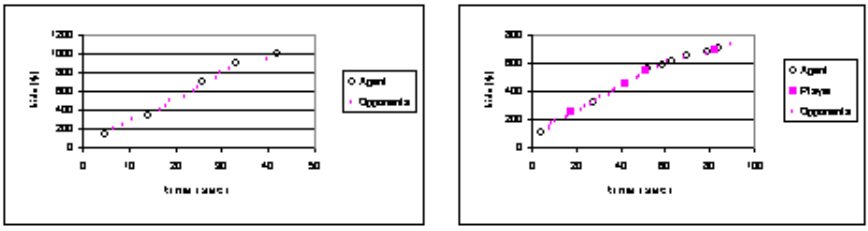


Fig. 4. (a) Optimal bidding; (b) Adjust update rate.

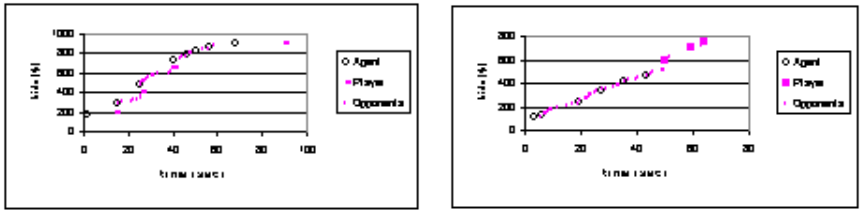


Fig. 5. (a) Adapt increment; (b) Jump bid — detect and quit.

4.1 English Auction

In an English auction, the participant has to decide whether or not to bid, how much to bid, and whether or not to abandon the auction (see the actions above). Observing the opponents (as in real life auction houses) is essential in taking such decisions, and by doing so, the participant gains two types of information: (1) how many bidders have dropped out of the auction (since they have lower valuations than the current bid); and (2) at what prices they dropped out [27]. The participant may also try to predict its opponents' behavior, and sometimes that means guessing their private information. Finally, it might be helpful to know if the opponents are *symmetric* (i.e., they use the same measurements to estimate their valuations), and if they have secret information about the item [28].

Optimal bidding in an English auction means bidding a small amount more than the current bid until you reach your valuation and then stop. This strategy, described by the rules in Figure 2, has the effect shown in Figure 4(a). Notice that the time is in seconds, but a real online auction might take a week or more. The winner (Agent in this case) is the one with the highest valuation (i.e., reserve price).

Since online auctions take place over a long period of time, it is hard to monitor them. The reasons are: (1) there is an Internet connectivity cost every time you enter a bid; and (2) there is an opportunity cost associated with logging

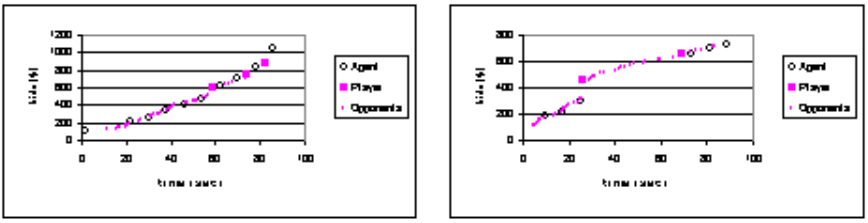


Fig. 6. Jump bid (a) detect and respond; (b) detect and wait.

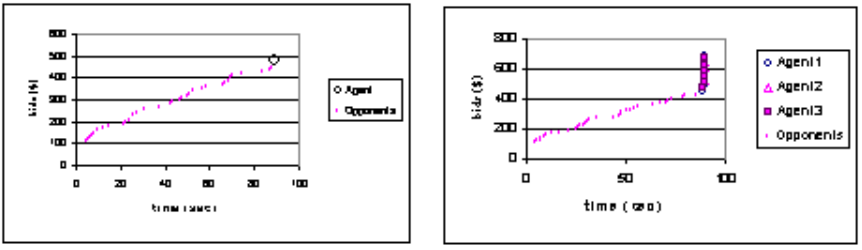


Fig. 7. (a) Snipe and win; (b) Sniping war.

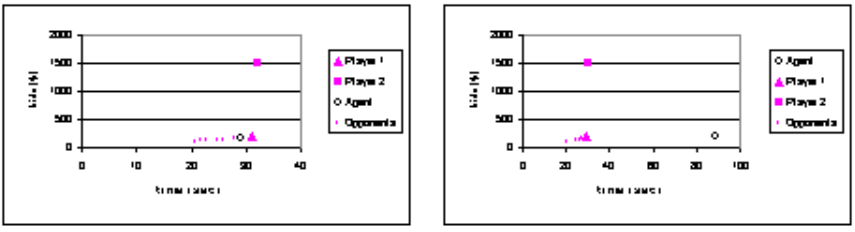


Fig. 8. Shielding (a) detect and drop; (b) detect and snipe.

on and entering the bid. An agent should optimize connections by connecting only when it is relevant to do so, and should be able to determine rapidly whether or not the auction suits its needs. Dropping out early means it can enter another auction early. Figure 4(b) shows Agent adjust its update rate to the average bidding rate. At the start of the auction, it makes few interventions, and as the activity increases, it participates more often (monitors the auction closely). Note that in this case, the agent makes bids every time it updates its information, but this is not an obligation.

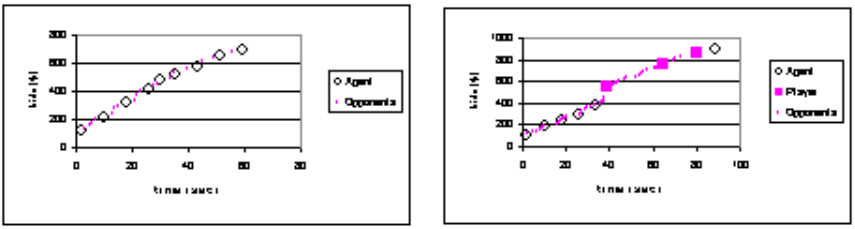


Fig. 9. (a) Increase valuation; (b) Jump bid – detect, wait, snipe.

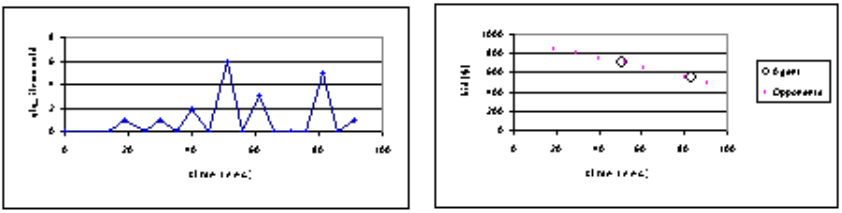


Fig. 10. Multi-item Dutch — safe buying.

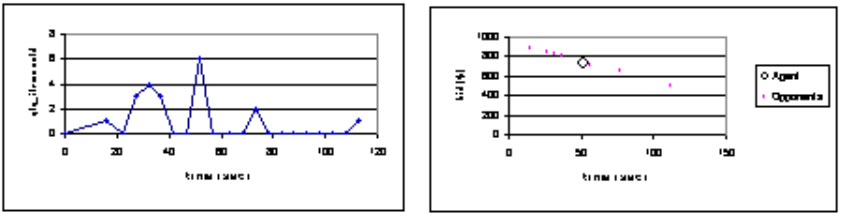


Fig. 11. Multi-item Dutch — panic buying.

In auctions where no minimum increment is enforced, we talk of *spontaneous bidding*. In this case, it might be useful to observe the bidding process and adapt one's increment to that of the opponents. Figure 5(a) shows Agent doing exactly that.

Jump bidding means entering a bid larger than necessary to send a signal to the opponents. The sender and the recipient of the signal may be better off in a jump bidding equilibrium: the sender saves bidding costs by deterring potential competition; and the recipient saves the costs of bidding against a strong opponent [29]. Figure 5(b) shows Agent detecting a jump bid made by Player and deciding to quit. Some opponents continue to bid but Player finally wins. In Figure 6(a), Agent responds with jump bids until Player (and everyone else) quits. In Figure 6(b), Agent detects a jump bid, and waits until the bidding goes back to normal before bidding again.

Sniping means waiting until the last minute of the auction, and trying to submit a bid, which barely beats the high bid and gives the opponents no time to respond. Obviously, if all bidders follow a sniping strategy, the game would become equivalent to a first-price sealed-bid auction, with all the bids submitted at the end. In Figure 7(a), Agent snipes and wins, and in Figure 7(b), three agents engage in a sniping war.

Bid shielding happens when a bidder puts in an early low bid (say $10) on an item, and then gets a friend (or a false identity) to put in an extremely high bid (say $500) on the same item. The high bid acts as a *shield* for the low bid, keeping anyone else from bidding. Just before the end of the auction, the bidder retracts the $500 bid, leaving the $10 bid as the winning bid on an item that should have gone for a higher price. Figure 8(a) shows Player1 and Player2 perform a shielding. Agent quits because the shield exceeded its valuation, and Player1 wins after Player2 retracts its bid. In Figure 8(b), Agent detects the shield and waits for it to be removed to snipe and win.

It can happen that a bidder reconsiders an item's valuation by observing how the opponents behave. Figure 9(a) shows Agent increase its valuation when its reserve price is met, the auction is about to close, and few participants remain. It changes the reserve price from 600$ to 900$ and stays in the game to win.

Finally, Figure 9(b) shows a combination of a several tactics. Agent avoids jump bidding by entering a waiting state, and at the end, it snipes and wins. This could be a way to hide your real intentions from your opponents.

4.2 Dutch Auction

When the market price of the auctioned item is a common valuation among bidders (e.g., an auction for personal computers), a bidder who wins the auction is the one with the highest yet possibly overrated valuation. This is called the *winner's curse*. In Dutch auctions, where a bidder selects a cut-off price at which to claim the item so long as no one else claims it, all participants try to avoid the winner's curse by *shading down* their bids slightly below their valuations (i.e., bidding less than what they think the object is worth) [27].

In Dutch auctions, no relevant information is disclosed in the course of the auction, only at the end when it is too late, which makes designing strategies a futile task. This is not the case with multi-item Dutch auctions where bidders watch the transaction prices, the number of remaining items (if available), the remaining time (if available), and decide whether to bid, wait for the price to drop, or quit. We designed a multi-item Dutch auction as follows: (1) identical items are put on sale and the unit price is made public, (2) as time passes, the seller decreases the unit price to generate interest, and (3) a buyer's bid is the quantity to purchase at the current price. The basic behavior of our agents is not different from that of the English auction. Instead of bidding a price, the agents bid a quantity. When created, an agent is given the quantity to buy, the minimum acceptable quantity, and the user's valuation.

We define a "safe buying" tactic as: "as the price decreases, when it reaches your valuation, buy the minimum quantity. Keep watching the auction, and

before it closes, buy the remaining items (possibly at a smaller price that your valuation).” The effect of this tactic is shown in Figure 10, where Agent (right) buys 4 items (minimum), and later, buys 3 more items to reach its maximum quantity of 7.

We define “panic buying” tactic as: “as the price decreases, if you see that a large number of items are sold, and your valuation is not met, then it might mean that your valuation is too low. Adjusting your valuation up will permit you to buy at least the minimum quantity.” In Figure 11, Agent (right) increases its valuation and buys the minimum 4 items needed. Notice that the risk of missing the minimum quantity is small, but the risk of the winner’s curse is high.

Finally, Figure 12 shows the effect of a the following “patient buying” tactic: “if the price drops fast, the buying rate is low, and you reach your valuation, then lower your valuation.” This tactic might save the buyer from the winner’s curse as it helps get the items at a lower price. There is however a risk that someone else buys all the items before the agent makes its move.

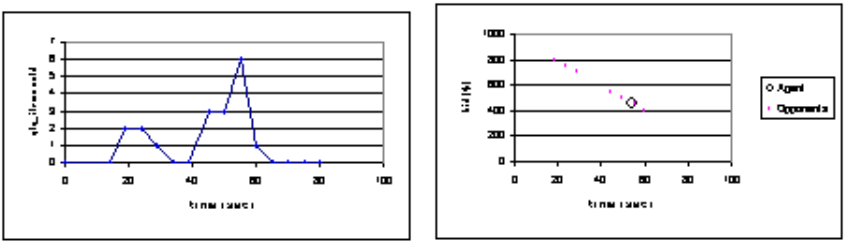


Fig. 12. Multi-item Dutch — patient buying.

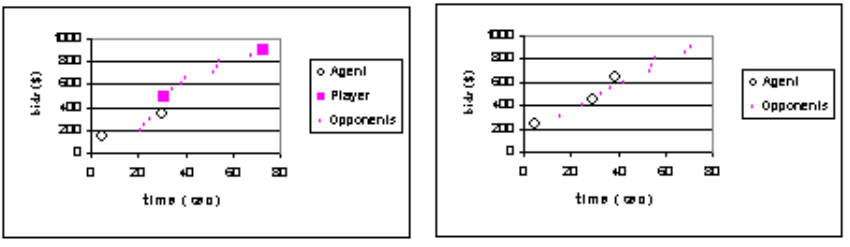


Fig. 13. Coordination — two English auctions.

4.3 Coordination

Using the same setting as before, we made several agents participate in separate auctions at the same time, and used the rule-based approach to manage their coordination. Here are some results.

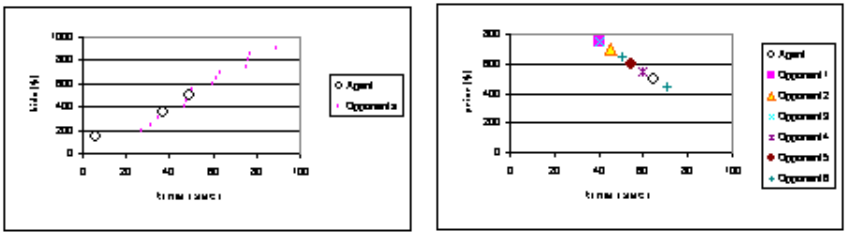


Fig. 14. Coordination — English and Dutch auctions.

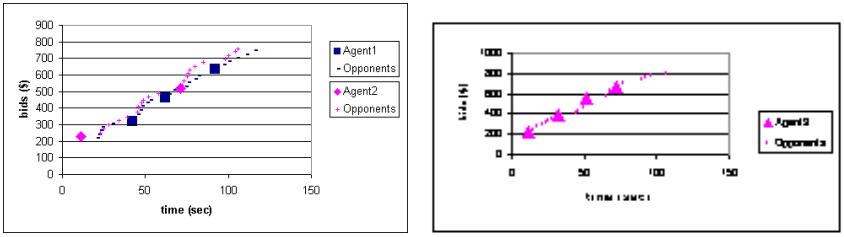


Fig. 15. Coordination — Agent3 quits, Agents 1 and 2 also quit.

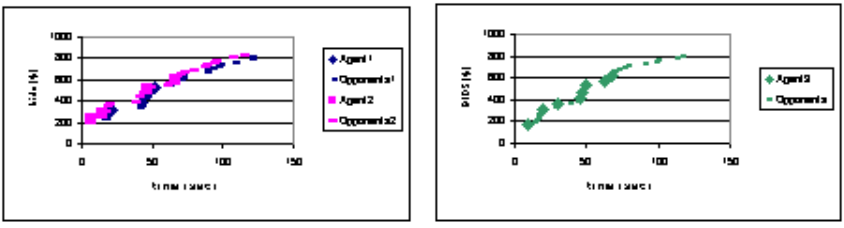


Fig. 16. Coordination — Agents 1 and 2 lose, Agent 3 quits.

Suppose we want two complementary items A and B, and we engage Agent1 in an English auction for A, and Agent2 in another English auction for B. The two agents negotiate separately, but they coordinate their actions using rules. Figure 13 highlights the following coordination scheme: "if Agent1 (left) detects a jump bid (i.e., the opponent is serious about winning, therefore Agent1 may lose its auction) then Agent1 must quit. In this case, Agent2 (right) must also quit to avoid exposure."

Now, suppose we want item A or item B, and we launch Agent1 in an English auction for A, and Agent2 in a Dutch auction for B. Figure 14 shows the effect of the following two coordination rules: "if the going price in the Dutch auction (right) is less than the current bid in the English auction (left), and our valuation is higher than the going price, we buy in the Dutch auction", "if the going price in the Dutch auction is less than the current bid in the English Auction, we quit the English auction."

For the following tests, three agents Agent1, Agent2, and Agent3 participate at the same time in separate (and independent) English auctions for items B, C and A respectively. The goal is to win "(B or C) and A", that is "B and A" or "C and A". Figure 15 shows Agent1 and Agent2 (left) bidding in parallel, and only one is leading at the same time while bids are always made in the cheapest auction. The figure also shows Agent3 (right) quitting (its reserve price is met), causing Agent1 and Agent2 (left) to quit. In Figure 16, both Agent1 and Agent2 (left) lose (their reserve price is met). At the same time, Agent3 (right) quits. Figure 17 shows Agent3 (right) snipe and win, and Agent1 and Agent2 (left) continue until Agent2 wins. In Figure 18, Agent3 (right) engages in a sniping war and loses to another sniper. Agent1 and Agent2 (left) have no choice but to quit.

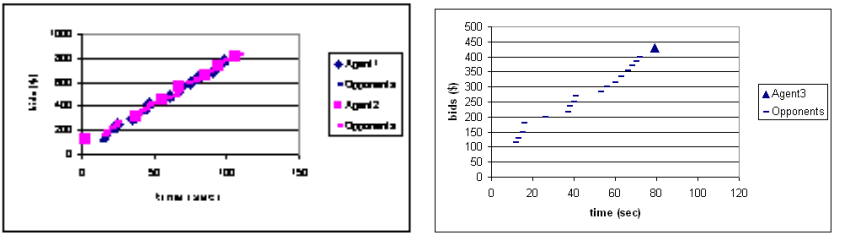


Fig. 17. Coordination — Agent 3 snipes and wins, Agents 1 and 2 keep going.

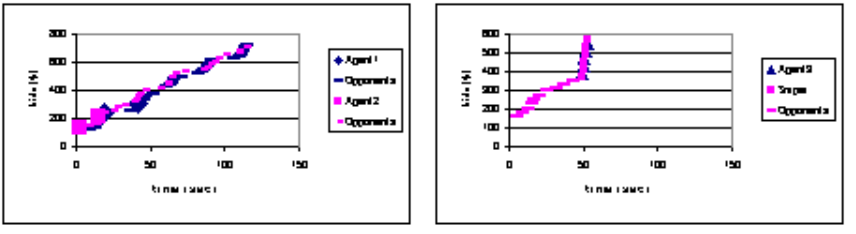


Fig. 18. Coordination — Agent 3 snipes and loses, Agents 1 and 2 quit.

5 Conclusion

The aim of the paper was to demonstrate that we could capture a wide range of bidding strategies and bid coordination schemes, using a rule-based approach, while supporting a wide spectrum of negotiation types. The well-known advantages of rule-based systems are the modularity and the uniformity (knowledge is represented using the same format). The possible inefficiency of rules and their eventual slow processing can be overcome by compiling the rules. Graphical tools

can also be used to browse the rule base in order to verify its coherence. Bid strategies usually involve fairly complex optimization algorithms and forecasting that could not be expressed directly as rules. We propose to make use of optimization algorithms and forecasting as procedural attachments in the action part of the rules.

Basic behavior of agents, various bidding tactics and coordination schemes across multiple auctions were implemented using our representation. They were tested in agent tournaments within simulated markets. So far, the results are encouraging and other possibilities are yet to be explored. As further points of interest we see the following. (1) The flexibility of this representation should be tested, and the limits of its expressiveness should be sought. (2) It is no secret that rules are a bit restricted in their ability to be adapted automatically, as agents are usually expected to adapt over time to improve their performance. It should therefore be investigated whether our approach can lend itself to machine learning techniques [14,20]. (3) We have been considering negotiations where the only negotiable attribute is the price, but in the case of a plane ticket, for instance, the attributes could be the price, the date of the flight, the airports (departure and destination), etc. Coordination across several multi-attribute negotiations should therefore be studied since it generates more complexity.

References

1. J. O. Kephart, J. E. Hanson, and A. R. Greenwald. Dynamic pricing by software agents. Computer Networks, 2000. To appear.
2. K. Jonkheer. Intelligent agents, markets and competition: consumers' interests and functionality of destination sites. Published Electronically on FirstMonday, 1999.
3. A. Moukas, R. Guttman, and P. Maes. Agent-mediated electronic commerce: An MIT media laboratory perspective. In proceeding of ICEC98, April 1998.
4. D. C. Parkes, L. H. Ungar, and D. P. Forster. Agent Mediated Electronic Commerce, chapter Accounting for Cognitive Costs in On-line Auction Design, pages 25-40. Springer-Verlag, 1999.
5. R. H. Guttman, A. G. Moukas, and P. Maes. Agent-mediated e-commerce: A survey. Knowledge Engineering 13(3), 1998.
6. A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In The First International Conference on the Practical Application of Intelligent Agents and Multi-agent Technology, April 1996.
7. M. Benyoucef, H. Alj, M. Vézeau, and R. K. Keller. Combined Negotiations in E-Commerce: Concepts and Architecture. Electronic Commerce Research Journal, 1(3):277–299, July 2001. Special issue on Theory and Application of Electronic Market Design. Baltzer Science Publishers.
8. T. Ito, N. Fukuta, T. Shintani, and K. Sycara. Biddingbot: A multi-agent support system for cooperative bidding in multiple auctions. In 4th Intl Conference on Multi-agent Systems (ICMAS-2000), Boston. MA, July 2000.
9. C. Priest. Algorithm design for agents which participate in multiple simultaneous auctions. Tech. Report HLP-2000-88, Hewlett-Packard, Bristol, England, July 2000.
10. C. Beam, A. Segev, and G. Shanthikumar. Electronic negotiation through internet-based auctions. Tech. Report 96-WP1019, UC Berkeley, December 1996.

11. C. Beam and A. Segev. Automated negotiations: A survey of the state of the art. Technical Report 97-WP-1022, Haas School of Business, UC Berkeley, 1997.
12. A. R. Greenwald and J. O. Kephart. Shopbots and pricebots. In16th Intl. Joint Conference on AI, volume 1, pages 506–511, Stockholm, Sweden, August 1999.
13. E. Gimenez-Fuentes, L. Godo, and J. A. Rodriguez-Aguillar. Designing bidding strategies for trading agents in electronic commerce. In Third Intl Conference on Multi-Agents Systems (ICMAS-98), Paris, France, July 1998.
14. W. Y. Wong, D. M. Zhang, and M. Kara-Ali. Negotiating with experience. In KBEM-2001, Austin, TX, 2000.
15. H. S. Nwana, L. Lee, and N. R. Jennings. Co-ordination in software agent systems. BT Technology Journal, 14(4):79–89, October 1996.
16. H. R. Varian. Economic mechanism design for computerized agents. In USENIX Workshop on Electronic Commerce, New York, NY, July 1995.
17. A.R. Lomuscio, M. Wooldridge and N.R. Jennings. A classification Scheme for negotiation in electronic commerce. In Agent-mediated e-commerce: a European AgentLink Perspective, pages 19–33, Springer-Verlag, 2001.
18. S. Y. Su, C. Huang, and J. Hammer. A replicable web based negotiation server for e-commerce. In 33rd International Conference on System Sciences, Hawaii, 2000.
19. B. N. Grosof. Courteous logic programs: Prioritized conflict handling for rules. Tech. Report RC 20836, IBM Research, T.J. Watson Research Center, May 1997.
20. D. Zeng and K. Sycara. Bayesian learning in negotiation. International Journal of Human-Computer Studies, (48):125–141, 1998.
21. C. McClintock and C. A. Berlioz. Implementing business rules in java. Java Developers Journal, May 2000.
22. D. M. Reeves, B. N. Grosof, Michael P. Wellman, and Hoi Y. Chan. Toward a declarative language for negotiating executable contracts. In Workshop on AI in Electronic Commerce, Menlo Park, CA, 1999.
23. B. N. Grosof, Y. Labrou, and H. Y. Chan. A declarative approach to business rules in contracts: Courteous logic programs in XML. In 1st Conference on Electronic Commerce, Denver, Colorado, November 1999.
24. ILOG JRules. `http://www.ilog.com/products/jrules`
25. N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. International Journal of Group Decision and Negotiation, 10(2), 2001. To appear.
26. M Berndtsson, S. Chakravarthy, and B. Lings. Coordination among agents using reactive rules. Tech. Report HS-IDA-TR-96-011, Skovde U., Sweden, October 1996.
27. L. J. Mester. Going, Going, Gone: Setting Prices with Auctions. Federal Reserve Bank of Philadelphia Business Review (March/April):3–13 1988
28. AGORICS. `http://www.agorics.com`
29. R. F. Easly and R. Tenorio. Bidding Strategies in Internet Yankee Auctions. Working paper, University of Notre Dame, 1999.

Distributed Transaction Management in a Peer-to-Peer Process-Oriented Environment

Theodore Chiasson, Michael McAllister, and Jacob Slonim

Dalhousie University, Halifax, Canada
{theo, mcallist}@cs.dal.ca, jacob.slonim@dal.ca

1 Introduction

As electronic commerce continues to permeate every aspect of our society, a significant segment of the population is being marginalized due to issues of accessibility. This is commonly referred to as the "Digital Divide" [4]. Social policy initiatives are attempting to increase accessibility through infrastructure enhancements and increased public points of presence. While this is helpful, it does little to address accessibility issues for persons who are functionally illiterate or cognitively impaired. Computerized systems will remain inaccessible to this population until the complexity of interacting with these systems is significantly reduced. The Knowledge-Acquiring Layered Infrastructure (KALI) project at Dalhousie University is attempting to reduce interaction complexity for this population through domain-specific personalization techniques that customize end-user interactions with computerized systems based on the abilities, preferences, and needs of individual end-users.

A paradigm shift from the client-server model to the peer-to-peer model of interaction would allow the end-user to maintain and control sensitive personalization information on their own computing devices. Interactions with the system could then be personalized in the context of the end-user's information, rather than exporting this information to remote servers for processing.

In the KALI project, a process-oriented peer-to-peer environment is used to manage end-user interactions in a domain-specific personalized fashion, where domains are defined as broad areas of interest (e.g., medical, grocery, entertainment) [3]. It appears that the process paradigm may facilitate some aspects of peer-to-peer application development [2]. However, due to the absence of global state from the process paradigm, it is not well-suited to traditional centralized approaches to distributed transaction management. This has forced us to take a new approach to the management of distributed transactions that adheres to the constraints of the process paradigm and does not require global state. This extended abstract describes our approach to managing distributed information in a process-oriented peer-to-peer environment without relying on a centralized transaction manager.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 182–192, 2002.

2 Background

Extensive domain-specific personalization information must be maintained in the system to effectively personalize interactions for functionally illiterate or cognitively impaired end-users. End-users will not allow the required personal data to be collected unless the information remains under their own control due to security and privacy concerns. Instead of releasing end-user personalization information, personalization of domain-specific interactions must therefore be performed at the end-user computing device.

Allowing the end-user system to act as a peer in the environment is difficult to achieve in a scalable manner due to the centralized or client-server architecture of most existing systems. A paradigm shift to the peer-to-peer model of computing is needed so that each participant in an interaction can act as a client and a server as the need arises.

It is difficult to construct fully distributed, scalable, peer-to-peer applications using traditional procedural and object-oriented languages and software development environments. These languages and development environments have evolved in the context of centralized and client-server computing. Single systems can be as large as millions of lines of code, with complex interdependencies among modules supported through constructs such as inheritance, templating, and type overloading. It is difficult to distribute the complexity of the resulting systems across many peers, due to the size of the resulting code base.

In contrast, the process paradigm as described in [8] appears well-suited to the development of peer-to-peer applications [2]. There is no concept of global state or global time in a process-oriented system, as all data and functionality are local to their owning process. This lack of shared state leads to process independence and allows for a peer relationship among processes, where each process can take on the role of client or server as need be. Thus, the process paradigm may be well suited to the development of scalable peer-to-peer electronic commerce applications [2].

In the process paradigm, software is constructed from processes with typed interfaces (also known as input and output ports). Each process encapsulates data and functionality, and processes can only be accessed through their typed interfaces. Thus, the input ports of a process entirely determine its interface. Processes communicate through message channels that connect the output port of one process to a type-compatible input port of another process. Processes exchange messages via typed input and output ports. A process does not, and indeed cannot, know about the implementation details of other processes.

As stated previously, we propose a shift from the existing client-server infrastructure to the peer-to-peer paradigm in order to maintain personalization information under the end-user's control. In the peer-to-peer paradigm, peer-to-peer interactions occur at all levels, including the application, operating system, and communications levels. Peer-to-peer computing is still in its infancy, however, and has mostly been applied only at the communications level. We perceive that changing to a peer-to-peer paradigm will involve changes in many facets of computing, as depicted in Figure 1.

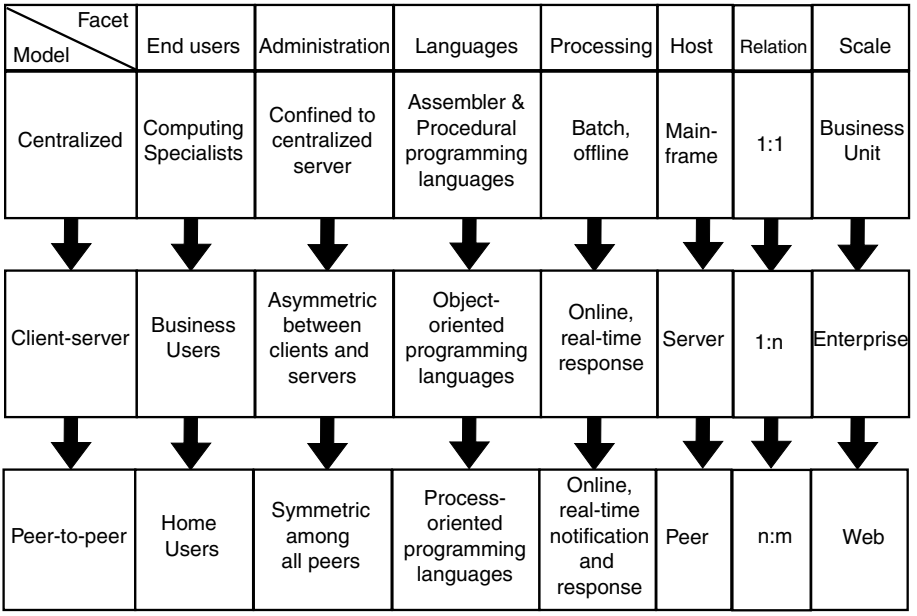


Fig. 1. Shifting Paradigms.

Typical electronic commerce interactions involve the execution of distributed transactions on electronic commerce servers in a client-server framework. In traditional distributed database transaction management implementations, global state is used to enforce the ACID properties (atomicity, consistency, isolation, and durability) on transactions. Distributed transactions that adhere to the ACID properties are guaranteed to be serializable, which ensures that they move the database forward from one consistent state to another consistent state. End-user interaction in the distributed transaction is limited to a request for the execution of a transaction and a response dictating the success or failure of the entire transaction based on the two phase commit protocol [5].

In the process paradigm, the use of the two phase commit protocol or other centralized concurrency management approaches would violate the independence characteristics of participating processes. In order to allow process independence and still support end-user ownership and control of end-user personalization information, we therefore anticipate the need for a process-oriented peer-to-peer distributed transaction management scheme that does not use global state [1]. In our approach, the end-user system is no longer a simple client. Using the fact that the end-user's system is a peer in the environment, we involve the end-user's processes in the management of distributed tasks. Further, we eliminate the need for a transaction manager by pre-allocating resources to eliminate conflicts and by executing the steps of a distributed transaction in a serial order instead of in parallel. The following section details this new approach.

3 Process-Relationship Approach

Due to the lack of global state in the process paradigm, we need a new approach to distributed data management that preserves distributed data consistency in the absence of a centralized transaction manager. Our approach, called the process-relationship approach, preserves distributed data consistency through the use of a distributed protocol involving three components: process relationship agreements (PRAs), micro-transactions, and tokens.

Prior to accessing remote services, a process must establish a PRA with each peer that will provide these services. The PRAs are business contractual agreements between peers for services; they can have either a long or short duration and are typically used for more than one interaction between the peers. Once a PRA is in place between two peers, micro-transactions can be executed under the auspices of the PRA. A micro-transaction is a distributed transaction that involves exactly two peers. Execution of a single micro-transaction effectively moves the distributed database forward from one consistent state to another consistent state. No centralized concurrency control mechanism is required because there are only two processes involved in the micro-transaction. Tokens are typed messages that are transmitted between peers. Different types of token are used during negotiation of PRAs, activation of micro-transactions, and signaling among peers.

The data structure for a PRA contains a static header section and a dynamic body section (see Figure 2). The header details the terms of the agreement and remains static for the lifetime of the contract. The body of the PRA is made up of containers, which consist of meta data, security and privacy restrictions, and data. Containers can be inserted, modified, or removed during the life of the contract. Two processes can be inconsistent with respect to one another at the business transaction level while a PRA is held between them, but must return to mutually consistent state before a PRA can be terminated. The PRA acts as a guarantee to both parties that their perceptions of one another's state will match before the PRA is terminated.

Tokens are used to set up and terminate PRAs and to facilitate all interprocess communication related to the management of distributed tasks. The structure of a token is shown in Figure 3. Tokens are used for the establishment of a PRA following a modified version of the contract-net protocol [7]. The task-announcement, bid, announced-award, and acceptance/refusal messages in the contract-net protocol map to the request, offer, contract, and response token types in the process-relationship approach, respectively. The protocol is extended with expiration criteria on the offers, allowing a contract token to be rejected if it is received after the corresponding offer token has expired. The exchange of tokens that occurs between two processes to set up a PRA is shown in Figure 4. Note that a request token does not commit a process to the acceptance of an offer, while an offer token commits a process to honoring a contract token meeting the terms of the offer.

In the process paradigm, all data access is local. The containers within a PRA provide a mechanism for storage of one process's data in a way that allows

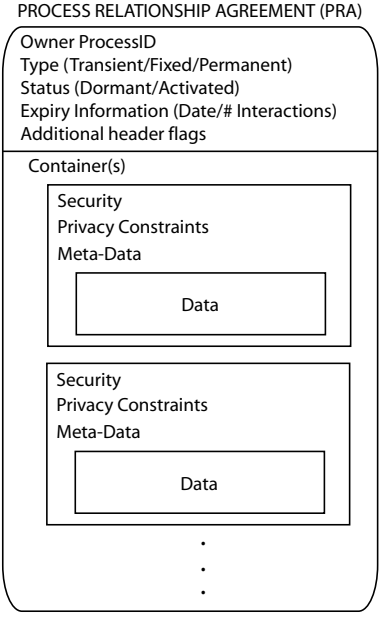


Fig. 2. Internal structure of a Process Relationship Agreement.

another process to access it. The owning process issues an activate token that allows the remote process to access specific parts of a container within a PRA. The results of this activation can be returned to the owning process in a response token. The final token type, signal, is used to query the status of an ongoing micro-transaction or to cancel a micro-transaction.

To efficiently support business transactions using the process-relationship approach, resources are pre-allocated before micro-transactions that lead to consumption of resources are initiated. As an example, a business transaction that involves the transfer of funds from an end-user's bank to a vendor will be broken into the following steps:

1. User process requests goods from vendor process
2. Vendor process requests funds from user process
3. User process requests pre-allocation of funds from bank process
4. Bank process responds to (3) with pre-allocation for user process
5. User process responds to (2) with payment info for vendor process
6. Vendor process requests pre-allocated funds from bank process
7. Bank process responds to (6) with transfer of pre-allocated funds to vendor process
8. Vendor process responds to (1) with goods for user process

If other concurrently running business transaction also require funds to be transferred from the end-user's account, the acts of withdrawing funds do not

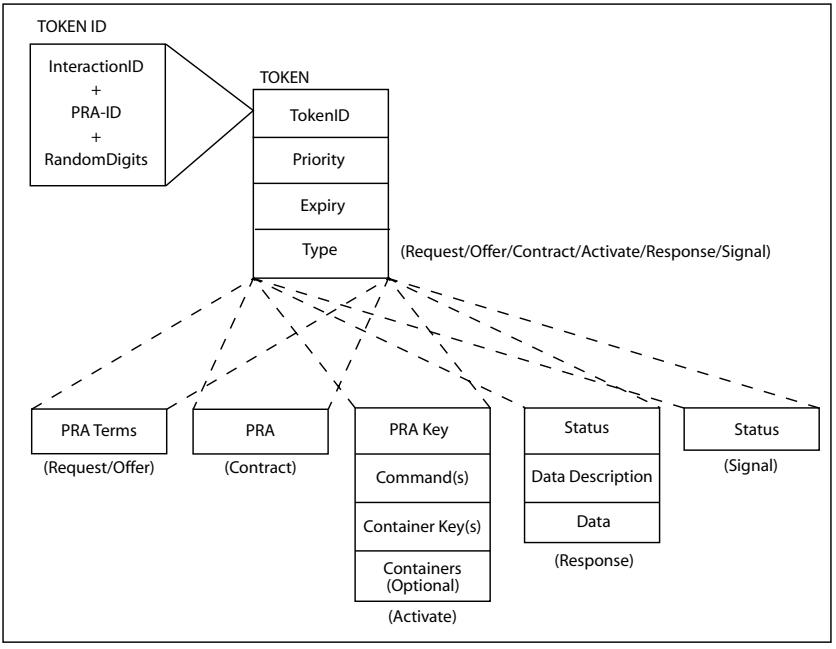


Fig. 3. Internal structure of a Token.

conflict because the funds are pre-allocated in all cases. The only conflict that remains is in the pre-allocation of funds, but this step is under the end-user's control. Thus, the end-user will not initiate the part of a business transaction that is outside of their control until they have guaranteed that it will not conflict with other concurrently executing business transactions.

Execution of a single micro-transaction involving two peers brings the state shared by these two peers forward from one consistent state to another consistent state. Each micro-transaction represents a consistent change in the overall distributed state of the system.

A micro-transaction can fail to complete if one of the two peers involved in the micro-transaction fails, or if the communications link between them fails. If the requesting peer fails, the peer executing the micro-transaction is free to commit and store the response message for later delivery (once the requesting peer has recovered). If the peer executing the micro-transaction fails, the local recovery mechanisms will ensure that the local state is preserved. For each micro-transaction that was in progress when a failure occurred, only the two peers involved in the micro-transaction are involved in the recovery. The only timing constraint that comes into recovery concerns the expiry of the PRA under which a micro-transaction is executing before a failed peer recovers.

It is possible that a PRA could expire or be cancelled before the business terms of the PRA have been met. While the distributed state of information will not be inconsistent from a database perspective, it could be inconsistent

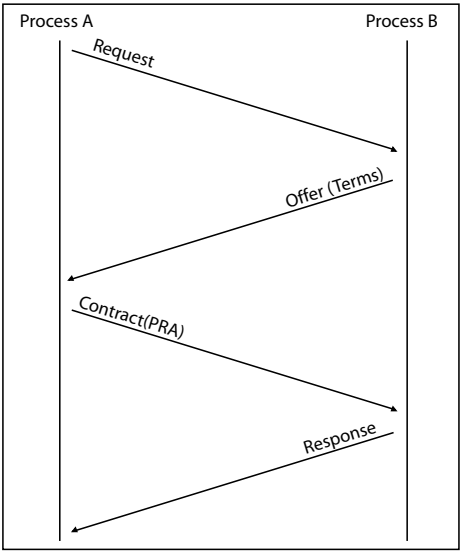


Fig. 4. Establishing a PRA.

with respect to the expectations of the parties that entered into the PRA. The PRA together with the micro-transaction logs executed under the PRA can help to determine if the obligations under the PRA have been fulfilled, and to determine any further required action. Treatment of the business effects of PRAs is independent of the underlying distributed information management approach taken, and is outside the scope of this paper.

4 Comparison of 2PC and PR Approaches

Figures 5 and 6 contrast our approach with the two phase commit approach using a travel reservation example. The precedence graphs included at the bottom of the figures highlight opportunities for concurrent execution in each of the approaches. The dotted lines connecting the termination of one step in the time line to the initiation of another step also indicate the precedence constraints.

The synchronization point in the process-relationship approach (Figure 5) is at the application level. While this synchronization point might be desirable for the end-user (so that hotel and car rentals are not booked without a plane ticket, for example), enforcement of this synchronization is not required to remain in a globally consistent state from a distributed database perspective. The synchronization point is important for the execution of appropriate business transaction, but does not impact upon the consistency of the distributed database. The entire distributed task is made up of 24 micro-transaction, each of which brings the global state forward from one consistent state to another consistent state.

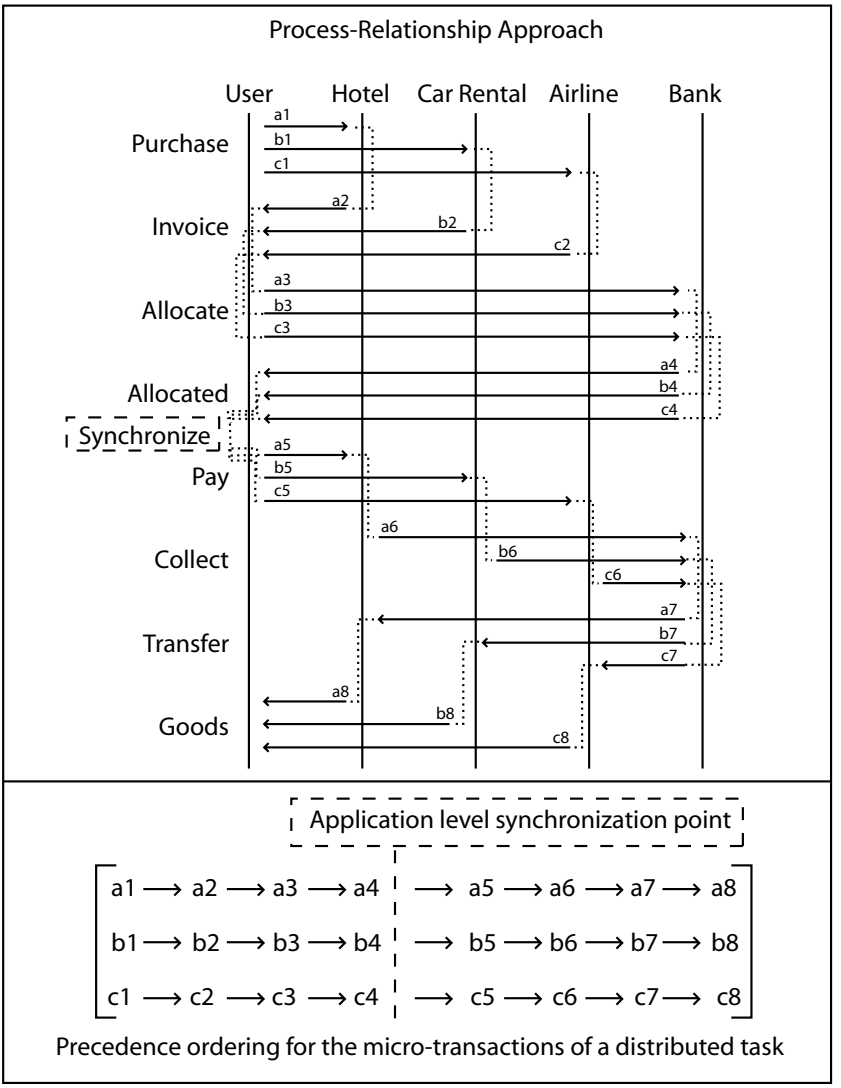


Fig. 5. Travel reservation example using the process-relationship approach.

All of the steps that make up the task are compensatable [6], since critical resources are pre-allocated and can therefore be de-allocated if the need arises. In the event that the end-user cancels the task, the steps that have already completed are compensated in the reverse order of their completion. Thus there is never a need to perform a rollback. Since the individual steps that make up the task do not conflict with respect to data accesses, there is no locking or blocking in the process-relationship approach. Furthermore, each step involves only one or two peers, eliminating the need for a global coordinator.

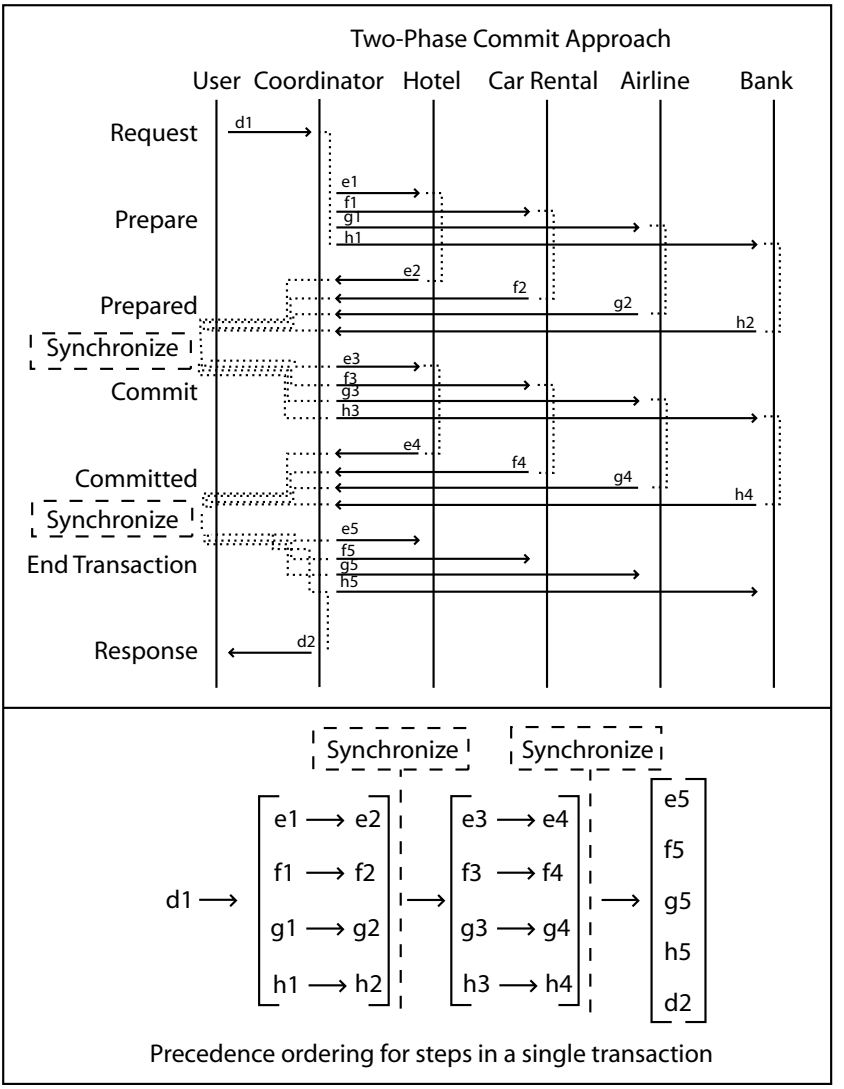


Fig. 6. Travel reservation example using the two phase commit approach.

The end-user process manages the initiation of micro-transactions that involve the end-user, allowing for a flexible response to problems. For example, if the car rental reservation fails to complete, the end-user can choose to initiate a car rental from another dealership and, if successful, continue processing the task with the new car rental in place of the old one. Thus, the progress accomplished on the airline and hotel reservations need not be lost.

In contrast, the two synchronization points in two phase commit approach (see Figure 6) are required by the two phase commit protocol to guarantee

globally consistent state is achieved, and the entire reservation task is a single distributed transaction that either commits or aborts. Furthermore, the end-user is a client in a client-server relationship, and the entire reservation process is executed as a single distributed transaction that either entirely commits or entirely aborts. Resources at each site are locked from the time that they are acquired until the transaction completes, potentially causing other concurrently executing transactions to block. If a problem arises during the execution of the distributed transaction, the entire transaction must be aborted or rolled-back, losing any progress made on the parts of the transaction that didn't encounter difficulties.

5 Conclusions

We have introduced a new approach to management of distributed information for a process-oriented peer-to-peer environment. This approach, called the process-relationship approach, adheres to the constraints of the process paradigm without compromising distributed database consistency. The key to the approach lies in placing the responsibility for dealing with resource conflicts on the owner of the resource in question. Forcing a process to pre-allocate resources to be consumed by a peer before initiating steps that cause the peer to request the resource effectively serializes the execution of business transactions. Each step in the business transaction is initiated by a token and executed as a micro-transaction.

Serialization of steps in a business transaction can lead to situations where, from a business transaction perspective, the distributed state is inconsistent. This situation is dealt with through the provision of process relationship agreements, which act as guarantees that domain-specific business transaction constraints will eventually be satisfied before the agreement terminates.

The process-relationship approach provides a mechanism for preserving data consistency and executing business transactions in a process-oriented peer-to-peer environment. Using this approach, it should be possible to store extensive personalization information under the end-user's control without divulging private information to other parties. Further research is required to investigate the generality, scalability, and performance characteristics of this approach.

Acknowledgments. This research is supported in part by grants from the Izaak Walton Killam Fund for Advanced Studies at Dalhousie University; MTT (an Aliant Inc. company); the Canadian Foundation for Innovation fund; and the National Sciences and Engineering Research Council of Canada.

References

1. G.K. Attaluri, D.P. Bradshaw, P.-A Coburn, N. Larson, A. Silberschatz, J. Slonim, and Q. Zhu. The CORDS multidatabase project. *IBM Systems Journal*, 34(1):39–62, 1995.

2. M.A. Bauer, N. Coburn, D.L. Erickson, P.J. Finnigan, J.W. Hong, P.-A Larson, J. Pachl, J. Slonim, D.J. Taylor, and T.J Teorey. A distributed system architecture for a distributed application environment. *IBM Systems Journal*, 33(3):399–425, 1994.
3. T. Chiasson, K. Hawkey, M. McAllister, and S. Slonim. An architecture in support of universal access to electronic commerce. *Journal of Information and Software Technology.*
4. P. Dickenson and J. Ellison. Getting connected or staying unplugged: The growing use of computer communications services. *Services Indicators*, 1st Quarter, 1999.
5. IBM. *Distributed Relational Database Architecture Reference.* IBM Distributed Data Library, 1990.
6. J. Moss. Nested transactions and reliable distributed computing. *Proceedings of the 2nd Symposium on Reliability in Distributed Software and Database Systems*, pages 33–39, 1982.
7. R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C–29(12):357–366, 1980.
8. R. Strom. A comparison of the object-oriented and process paradigms. *SIGPLAN Notices*, 28(4):88–97, 1986.

Using XML Schemas to Create and Encode Interactive 3-D Audio Scenes for Multimedia and Virtual Reality Applications

Guillaume Potard and Ian Burnett

Whisper Laboratory, University of Wollongong, Australia
{gp03, ian_burnett}@uow.edu.au

Abstract. An object-oriented 3-D sound scene description scheme is proposed. The scheme establishes a way to compose and encode time-varying spatial sound scenes using audio and acoustical objects. This scheme can be used in applications where efficient coding of interactive 3-D sound scenes is needed (e.g. interactive virtual displays and videoconferencing). It can also be used in non-interactive application such as cinema and 3-D music. The scheme offers clear advantages over multi-channel 3-D sound formats regarding scalability and interactivity with the sound-scene because each object has its own set of parameters and can be modified by the end-user at the decoding stage. The object-oriented approach also allows the creation of macro-object descriptors that allow fast and efficient coding of 3-D sound scenes using references to macro-object libraries. The scheme has been implemented in a XML schema and can be used to define 3-D sound scenes in XML format in a standard way.

1 Introduction

Multimedia displays often suffer from a lack of realism in the audio representation. Usually, the audio component is only a pale representation of the three-dimensional sound fields that surround us everyday. By using three-dimensional sound, important cues and realism can be added to the multimedia or virtual reality presentation [1].

We propose an XML [2] based scheme which can be used to create, encode and transmit user-modifiable 3-D sound scenes over the World Wide Web. The scheme can be used in various ways:

- Full description of sound scenes (temporal and spatial information), this suits 3-D music, cinema and broadcasting applications.
- Partial description of sound scenes (structural and hierarchical information only), the objects parameters such as position of objects being set by the end-user. This suits more interactive applications such as virtual-reality, video-conferencing and video games.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 193–203, 2002.

The scheme presented here adopts an object-oriented architecture in order to reflect the fact that natural 3D sound scenes are normally formed by the interaction between physical objects such as: sound sources, reflecting and obstructing objects, reverberant spaces and a medium. To describe scene thoroughly, the scheme encompasses all these objects plus the description of temporal events and the change of object parameters over time (e.g. displacement). The scheme also defines the hierarchical relationships between the sound objects. This is to address the fact that complex sound objects are usually made of several individual sound objects (for example, a car has a reflective body and emits multiple sound: exhaust, engine noise, tyre friction, horn etc.). Groups of objects are called macro-objects and they are explained in part 2.3.

XML has been chosen to implement our scheme since it is a well-established description language and goes well with the object-oriented approach of the scheme. The scheme was implemented in a XML [2] schema and consequently, any represented 3-D sound scene will be an XML file built upon this schema.

Compared to traditional multi-channel 3-D sound formats such as 5.1 [3], the scheme proposed here offers a flexible solution to composing and transmitting virtual 3-D sound scenes over the World Wide Web. Indeed, the object-oriented approach allows real-time access and modification of sound objects and scene parameters. Scenes can also be composed algorithmically, making the encoding of sound scenes very efficient.

This document will first present the composition of 3-D sound scenes and their implementation and applications in the World Wide Web.

2 Composition of 3-D Sound Scenes

The scheme describe sound scenes using four kinds of descriptors:

- Elementary Objects
- Structural and hierarchical (Environments and macro-objects)
- Temporal (Choreography)
- Control (Scene commands)

2.1 Elementary Objects Descriptors

The elementary objects are the basic components of the 3-D sound scene scheme. They are the representation of physical objects in real 3-D sound scenes. The schemes encompasses four types of elementary objects:

Sound Source: A sound signal input in the virtual acoustic space.

Reflective/Obstructing surface: A virtual surface that reflects, absorb and obstructs the sound field.

Medium: Defines propagation speed and frequency dependant attenuation of the signal during its propagation.

Listener: Defines the position and orientation of the listener in the virtual environment.

Room: Defines acoustical properties of the room (reverberation time, etc.)

Each of these elementary objects can have a set of parameters and a suggested set is listed in Table 1 below.

Table 1. Parameters of the elementary objects.

OBJECTS	ATTRIBUTES
Sound Source (active)	Position, Orientation, Directivity, Spatial size and shape, resource address (URL or filename)
Obstructing/Reflective (passive)	Position, Orientation, Geometry, Reflecting and absorbing coefficients
Medium	Attenuation coefficient and speed of sound (function of Temperature and Humidity)
Listener	Position, Orientation
Room	Reverberation time

2.2 Environment Descriptors

Environments are a grouping mechanism that list all objects being present in the same acoustical space. Only the objects of the same environment will interact with one another. This allows applying different acoustical effects to the objects and thus using different acoustical effects in the same scene.

2.3 Macro-Objects

In analogy to concrete physical objects, *3-D Sound Objects* are also organised into a hierarchy. For instance, a moving car has also its children objects such as tyres and exhaust moving at the same speed. Hence, mechanical relationships between physical objects also exist in the sound domain.

For the purpose of encoding sound scenes it is thus interesting to express the relationships between the elementary objects. Therefore, if one wants to move an object, all its child objects will move automatically, this simplifies greatly the description of temporal events and parameter changes.

In the scheme, macro-objects can be created at initialization time and during run time (if relationships between objects need to be modified while the scene is playing).

Macro-objects offer another important advantage: the possibility to create libraries of macro-objects (Section 2.3).

Example of Macro-object. Figure 2 illustrates the concept of macro-objects. In this example, a choir is defined by a macro-object. It is, in turn, composed of several lower-level singer macro-objects. A singer object is the linkage of a sound source with particular directivity (mouth) and a reflecting/absorbing surface representing the singer's body.

Other examples of macro-objects are: a Jazz band, an automobile, a speaker, a crowd etc.

There are several possible ways of creating and utilising macro-objects:

- Grouping of several elementary and/or macro-objects
- Repetition (cloning) of the same object with a position change (e.g. a swimming pool atmosphere can be encoded very efficiently by using one "splash" sound repeated many times over a surface).
- Repetition of an object after some transformation. (e.g. a group of choristers can be created from the same singer-object repeated several times and a pitch transformation applied).

Hence, complex 3-D audio scenes can be encoded efficiently using high-level object descriptions.

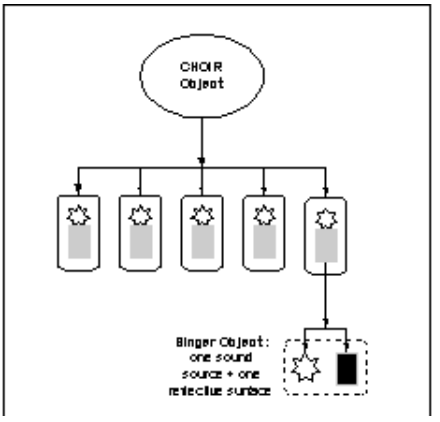


Fig. 1. Example of macro-object describing a choir.

Macro-Objects Libraries. The library contains a high number of macro-objects and is shared between the encoder and decoder both to compose and interpret the scene (Figure 2). Hence, because the macro-object's description can be found in the library it is not necessary to transmit all the child objects parameters but only few parameters of the macro-object, thus, if the library is extensive, this reduces significantly the amount of information transmitted and simplifies the description of scenes.

If the description of the macro-object cannot be found in the decoder library, the decoder can fetch the description from the encoder and add it to its local library.

An XML schema has also been developed [14] to create macro-objects. Therefore a micro-object library is a collection of XML files representing macro-objects.

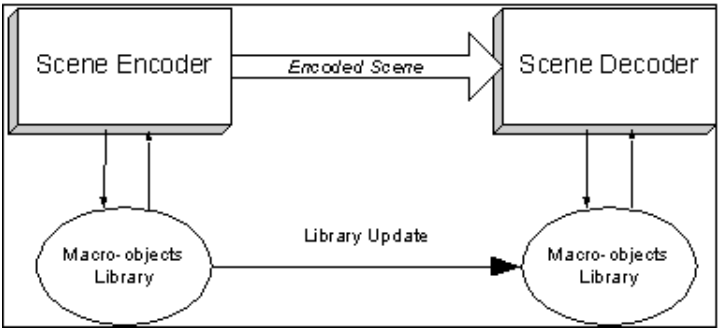


Fig. 2. Common Library of macro-objects used by the scene encoder and decoder.

2.4 Command Descriptors

These are special commands issued to the decoder. These commands are issued to control the decoding and rendering of the scene (e.g. scene refresh rate, Doppler effect on or off etc.)

2.5 Temporal Descriptors

Description of temporal events and parameters are used to describe the temporal behaviour of the scene. The temporal descriptors are placed in a score, in analogy to music composition. The score defines chronological events when objects are appearing, disappearing, moving and when object parameters are changing. Any data of the sound scene (sound objects parameters, macro-objects, commands) can be changed and modified by the score. Figure 3 depicts a sound object with changing azimuth and elevation values.

Scene Score Format. The score contains two parts:

- An initialisation part: is used to set attributes and parameters of the scene before the scene being rendered, this is also used to set acoustic environments and macro-objects seen in Sections 2.2 and 2.3.
- A performance part: contains the parameters changes such as object displacement, etc.

The *Initialisation* and *Performance* scores have formats as depicted in Figure 4.

The fields of the score are explained below:

Command: The action to be taken on the object (operation)
Object: The object(s) which the action is applied upon (operand)

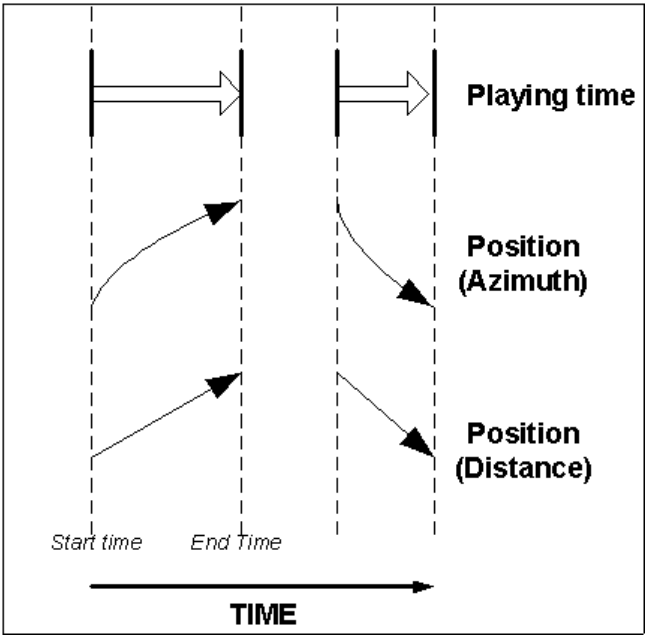


Fig. 3. Example of score description of a moving sound object.

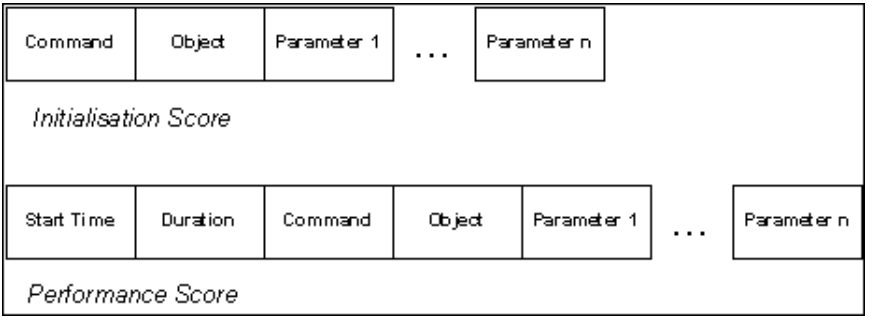


Fig. 4. Fields of the Initialisation and Performance score.

Parameter: A field containing additional parameters used by the command (e.g. destination position of the object for a move command), there can be an unlimited number of parameters.

Start Time: Time at which the action should be taken (therefore does not exist in the initialisation score).

Duration: Duration of the command being applied.

Therefore each action and temporal event is represented by one *line of score*.

Example of Command in the Score:

```
0   5   Move_To   Object_3   5   3   -2
```

This line of score describes the displacement of `Object_3` from its current position to coordinates (5,3,-2) in 5 seconds starting at 0 second.

New commands can be added to the scheme, as long it is recognised by the decoder. Therefore, if system upgrade is performed in the future, there is no need to modify the XML schema and previously XML encoded scenes.

2.6 Example of Scene Composition

Figure 5 illustrates the configuration of an audio scene. This scene contains two *acoustic environments*. The first environment contains two *sound sources*, two *reflective surfaces*, a *medium*, a room and a *macro-object* describing a speaker. Hence the sound sources will be reverberated and reflected by the objects present in the same environment.

The second environment contains only one sound source, and no reverberation or acoustical effects will take place since the source is alone in its environment. The total 3-D scene will be formed by the mix of the two environments and will be controlled by the *score*, the *Commands* and the *listener* objects.

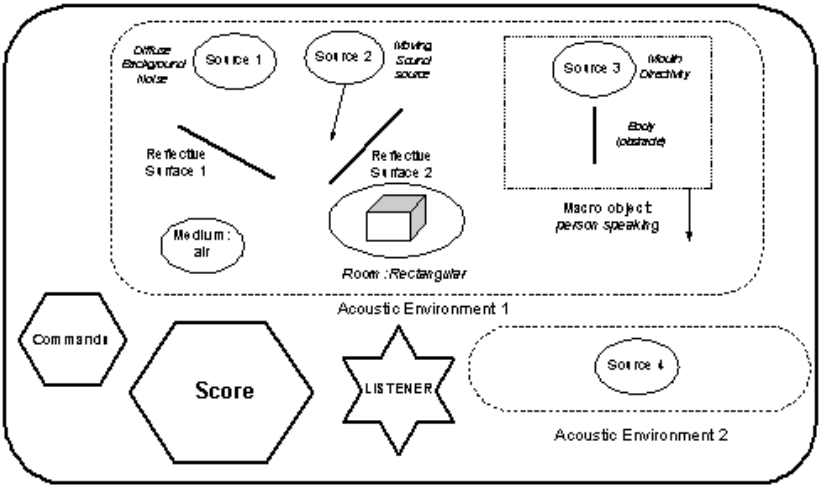


Fig. 5. Example of configuration of an overall 3-D sound scene containing two acoustic environments, several objects and macro-objects.

3 Implementation and Applications of the Scheme

The 3-D sound scene description scheme can be used in many applications such as cinema 3-D Sound, videoconferencing, virtual reality, interactive web-pages etc..

3.1 Example of Implementation in Web-Browsers

The scheme can be used in many situations; Figure 6 shows an example of use of the scheme in a system delivering interactive 3-D Sound along with a Web page. The 3-D sound scene decoding system can be implemented into one Java applet [4] that parses the XML data, downloads the necessary resources (such as sound signals) and carries out the signal processing tasks in order to spatialise the sound. The Java applet can also provide an interface to the user to allow him/her to modify the scene in real-time. This interface can have varied forms and is designed depending on the application and on the level of interactivity wanted (for example: in a video-game type of application, the scenes parameters will be modified by the user actions, whereas for a broadcasting application, the scene parameters need not be modified).

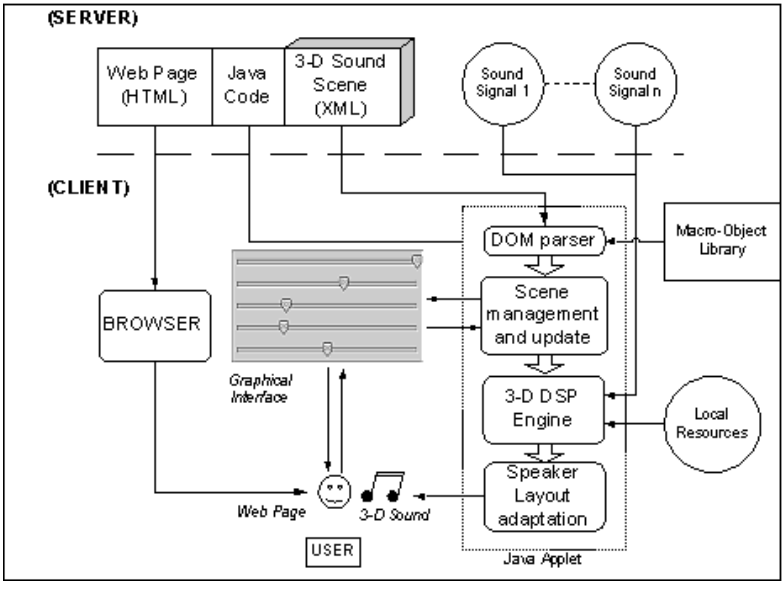


Fig. 6. Example of utilisation of the 3-D sound scene scheme with Java applets and Web-browsers.

The present example shows how the system can be implemented in Java applets and web-browser, but it can alternatively be implemented in a browser

plug-in, in a standalone program, or as a dynamic library shared by other programs.

3.2 The Scene Management and Update Unit

The *Scene management and update unit* performs several tasks:

- Through the DOM Interface [5], it reads the scene description.
- It drives the 3-D DSP engine according to the scene description and event score.
- It updates the scene and objects parameters.
- It collects commands from the user, and outputs results.
- It makes decisions. For example, it can decide which object should be included in the scene depending on the object's priority level and the resources available at the decoder.
- It takes actions according to events happening in the scenes (e.g. collision of objects).
- It performs the acquisition of online and local resources such as sound signals and macro-object definitions.

3.3 The 3-D Digital Signal Processing Engine

The 3-D engine, controlled by the scene management unit, carries out the DSP tasks in order to spatialise the sound signals and recreate the acoustical "feeling" of the virtual environment.

Each sound source is processed depending on its own parameters and on the parameters of the acoustic objects present in the environment (medium and reflecting surfaces objects). The process applied to the sound source also depends on the scene commands (Section 2.4) and scene score (Section 2.5).

Other factors such as available processing power can also be taken into account (i.e. to allow scalability).

After individual processing, the processed objects are summed together to form the total 3-D sound scene (Figure 7).

Types of Processing Applied to the Sound Signals. The sound signals are usually "dry" (non-reverberated) and monaural. In order to spatialise the sound signals, they are to be transformed by different signal processing tasks:

Spectral Filtering: to simulate medium attenuation and signal attenuation when bouncing against absorbing surfaces occurs.
Image or ray-tracing algorithm: to compute the early reflection response of the room (specular reflections) [6].
Reverberation: to simulate the diffuse (late) reverberation of the room [7].
Delay: to account for delay due to distance of objects.
Pitch shifting: to simulate Doppler effect with moving objects.

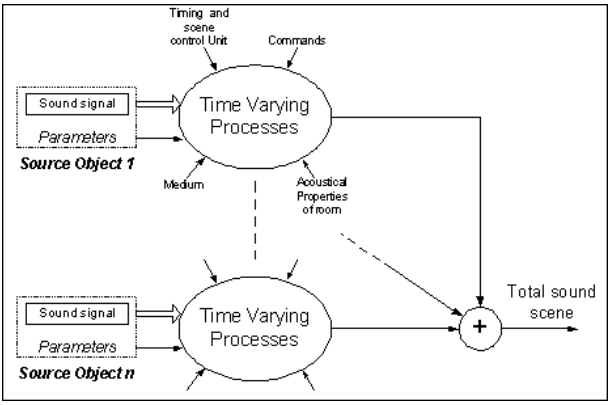


Fig. 7. Overview of the signal processing configuration

Spatialisation: to place the sound objects in particular spatial positions. Depending on the reproduction system, the processes involved are: binaural [8] (for headphones), transaural [9] (for two speakers) and multi-speaker panning such as Ambisonics [10] or VBAP [11].

3.4 Creation of Scenes

Depending on the application, 3-D sound scenes can be created automatically (e.g. from a 3-D visual content), semi-automatically or manually. Figure 8 shows how sound scenes can be created manually with *scene composer software*. The scene author composes the scene through an explicit graphical interface. In turn, the DOM writer generates the XML file according to the 3-D sound scene XML schema. Therefore, the scene author can be completely oblivious to XML.

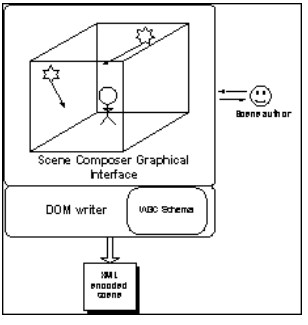


Fig. 8. Scene composer software generating the XML-coded 3-D sound scene.

4 Conclusion

An object-oriented scheme for describing time-varying spatial sound scenes has been proposed. The scheme has been implemented in a XML schema and can be used in many applications where 3-D sound is required (cinema and radio broadcasting, interactive Web pages and displays, virtual reality, videoconferencing, human-machine interfaces, etc.)

The object-oriented chosen to develop the scheme allows real-time control of the 3-D sound scene and scalability. The object-oriented approach also enables the creation of macro-objects that can be stored into libraries and used in scene descriptions; reducing the amount of transmitted data and simplifying the encoding of sound scenes.

There is ongoing effort at the University of Wollongong to integrate the 3-D sound scheme into Virtual Reality languages such as Web3D [12] and multimedia coders such as MPEG-4 [13].

The scheme as well as some examples of XML encoded 3-D sound scenes can be downloaded from [14].

References

1. Begault, D.R., 3-D sound for virtual reality and multimedia, Academic Press, 1994.
2. World Wide Web Consortium: `http://www.w3c.org`
3. Bosi, M., Multichannel audio coding and its applications in DAB and DVB, International Conference on Communication Technology Proceedings, 2000. WCC–ICCT 2000, Vol. 2, 2001.
4. Sun Java home page: `http://java.sun.com`
5. World Wide Web Consortium: `http://www.w3.org/DOM/`
6. Vorländer, M., Simulation of the transient and steady-state sound propagation in rooms using a new combined ray-tracing/image-source algorithm, Journal of the Acoustical Society of America, Vol. 86(1), July 1989, pp 172–178.
7. Moorer, J.A., About This Reverberation Business, Computer Music Journal, Vol. 3(2), pp 13–28
8. Begault, D.R., Binaural Auralization and Perceptual Veridicality, Proceedings of the 93rd Audio Engineering Society, San Francisco, October 1992, Preprint 3421 (M-3)
9. Bauck, J. and Cooper, D.H, Developments in Transaural Stereo, IEEE workshop on Applications of Signal Processing to Audio and Acoustics, 1993, pp 114–117
10. Gerzon, M.A, Ambisonics in Multichannel Broadcasting and Video, Journal of the Audio Engineering Society, Vol. 33(11), November 1985, pp 859–871.
11. Ville Pulkki, Virtual Sound Source Positioning Using Vector Base Amplitude Panning, Journal of the Audio Engineering Society, Vol. 45(6), June 1997, pp 456–466
12. Web 3D Consortium: `http://www.web3d.org`
13. Overview of the MPEG-4 standard: `http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm`
14. Whisper Laboratory Web page: `http://www.whisper.elec.uow.edu.au/guillaume.htm`

The Design of High-Level Database Access Method in a Web-Based 3D Object Authoring Tool*

Boon-Hee Kim[1], Jun Hwang[2], and Young-Chan Kim[1]

[1] Chung-Ang University, Computer Science and Engineering,
221, Huksuk-Dong, Dongjak-ku, Seoul, 156-756, Korea
bhkim@sslab.cse.cau.ac.kr
[2] Seoul Women's University, Computer Science and Engineering,
Seoul, 139-774, Korea

Abstract. Because of rapid expansion of computing power and the supply of high speed network technology, we want a realistic service like Virtual Reality on the Internet. At present, the majority of the services of virtual reality on the Internet use VRML(Virtual Reality Markup Language). But existing integration of VRML with a Database do not supply functions which integrate with other programming languages. And it is also complicated to reconstruct Virtual object whenever it changes. Also the VRML do not have the function of using other data formats in conjunction with VRML.
In this paper, we design and implement an authoring tool based on VRML and XML(Extensible Markup Language) that can the creation and modification of VRML and XML object on a high-level(that is to say, users of this authoring tool need not usage of DBMS). Existing VRML and XML authoring tools do not have the function of connection to the database so they must use Embedded SQL or Server Side Include where developers must insert database code into VRML and XML code. That is, existing Virtual Object(VRML and XML Object) authoring tools isn't convenient to user of the authoring tool. In this paper we propose an authoring tool that integrates Virtual Object and database so that a 3D e-Catalog based on Virtual Object connection to DB can be made. Therefore developers are offered the ability to manipulate Virtual Object on a high-level.

1 Introduction

Recently the scale of e-commerce has tremendously increased which in turn has increased the interests in e-Catalogs that offer information about products. Also reality-like 3D environments based on high-computing power and high-speed networking technology like virtual museums or virtual shopping malls are served on the Web. These are mostly developed with VRML(Virtual Reality

* This work was supported from 2001 industrial-educational cooperation of small & medium business administration.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 204–214, 2002.

Makeup Language) [1] which is a representative 3D representation language. Cosmo World [2], Internet Space Builder [3], Spazz3D [4], etc. are commonly used authoring tools. In current methods while creating a Virtual Object object connected to a database, Virtual Object authoring tools has some problems — users of this authoring tool must modify source code.

In this paper the 3D e-Catalog authoring tool proposed solves this problem. Also in this implementation, a Virtual Object translator is implemented to dynamically control Virtual Objects and a Virtual Object code extractor is implemented to make various Virtual Objects translated through the Virtual Object translator to an e-Catalog Virtual Object file. Therefore this tool offers high-level Virtual Object manipulation to developers.

This paper is organized as follows : problems of existing Virtual Object authoring tools connection to database and the proposed solution is described in Section 2. The design and architecture of this tool is described in Section 3. The implementation of the tool is described with an example in Section 4. The evaluation of methods is described in Section 5. Conclusion and future work are presented in Section 5.

2 Existing Virtual Object Authoring Tools

Existing Virtual Object authoring tools are focused on making Virtual Objects and do not supply functions to integrate other programming languages to itself. Accordingly, they cannot automatically make a connection between Virtual Object and database so they must use Embedded SQL or Server Side Include that must insert database code into Virtual Object code.

But these methods have the following problems. First, there is only a passive method for the database control because SQL statements are inserted into Virtual Object code. Therefore it is difficult to dynamically control Virtual Objects because Virtual Object code must be modified in order to do SQL programming. Second, it is possible that someone decompiles a java class to find out the user name and password, used for connection to the database. This is because the all methods of Embedded SQL and Server Side Include are established in a java class.

This paper provides the following solutions for these problems. First, Data Input Application implemented in this paper provides an easy connection method to the database without developers modifying the Virtual Object code. Second, the problem of security is solved as the SQL statements are run on the server using EAI and Client/Server Model when Virtual Objects connect to the database. A Virtual Object translator is implemented so that Virtual Object files can be made into Virtual Objects that can be controlled by the authoring tool. So a Virtual Object code extractor is implemented so that various Virtual Objects can be made into one Virtual Object e-Catalog file.

According to these solutions, the authoring tool implemented in this paper can be used to make a 3D e-Catalog for e-Commerce on the Web.

3 The Design and Architecture

3.1 The System Outline

The 3D e-Catalog authoring tool based on Virtual Object presented in this paper is composed of a database storing product information and product Virtual Objects, a development applet for creating a 3D e-Catalog, a customer applet for showing the 3D e-Catalog to customers and a server with SQL statements for communication between applets and the database. Figure 1 shows the system overview. The database and server communicate via JDBC for data transfer. The applets and server communicate via RMI. The problem of security and easy connection method to the database without developers' modifying the Virtual Object code is solved as the SQL statements are run on the server using EAI and Client/Server Model when Virtual Objects connect to the database.

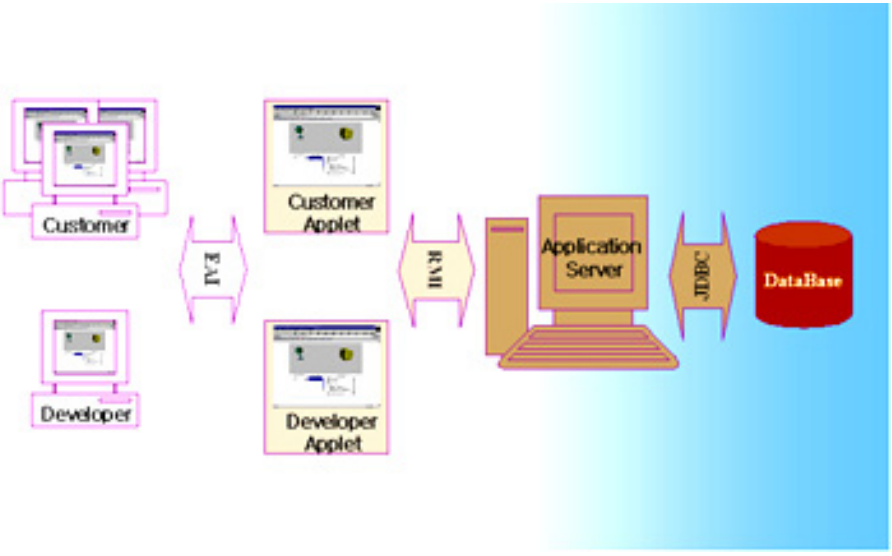


Fig. 1. The System Overview

3.2 Data Input Application

In order to create an e-Catalog, the database must have product information, product Virtual Objects and information that is referenced between product information and product Virtual Object files. Data Input Application has functions that get Virtual Object files and product information entered by user, and insert them into the database with the product identity entered by the user, which distinguishes it from others. This product identity becomes a primary key in the

product information database and the product Virtual Objects database. After the front of the product identity is appended with Virtual Object comments it is inserted into the Virtual Object file header and stored on database.

3.3 The Development Applet

The development applet is the most important part of the authoring tool proposed in this paper. It has functions that manipulate various Virtual Objects that are taken from the database and make into one Virtual Object file that is used as the e-Catalog.

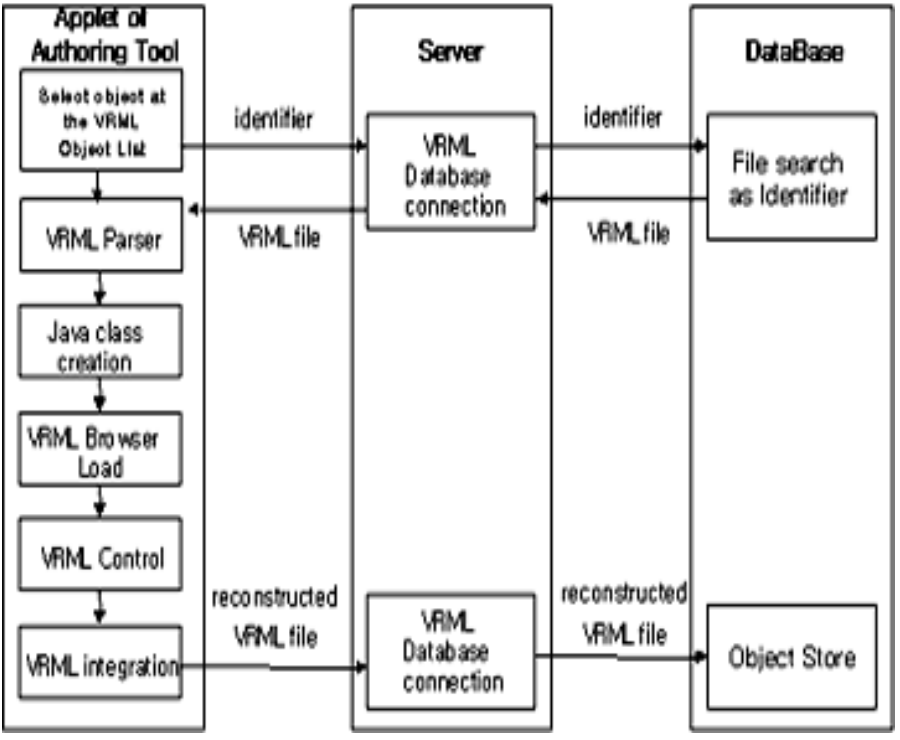


Fig. 2. Flowchart for the execution of Authoring Tool Applet

The structure of the Development Applet. The development applet consists of four parts. : A function for Product information and product Virtual Object file retrieval from the database, a Virtual Object translator that makes a Virtual Object from a Virtual Object file to be manipulated, a user interface to manipulate Virtual Objects and a Virtual Object code extractor to make various Virtual Objects into one Virtual Object file.

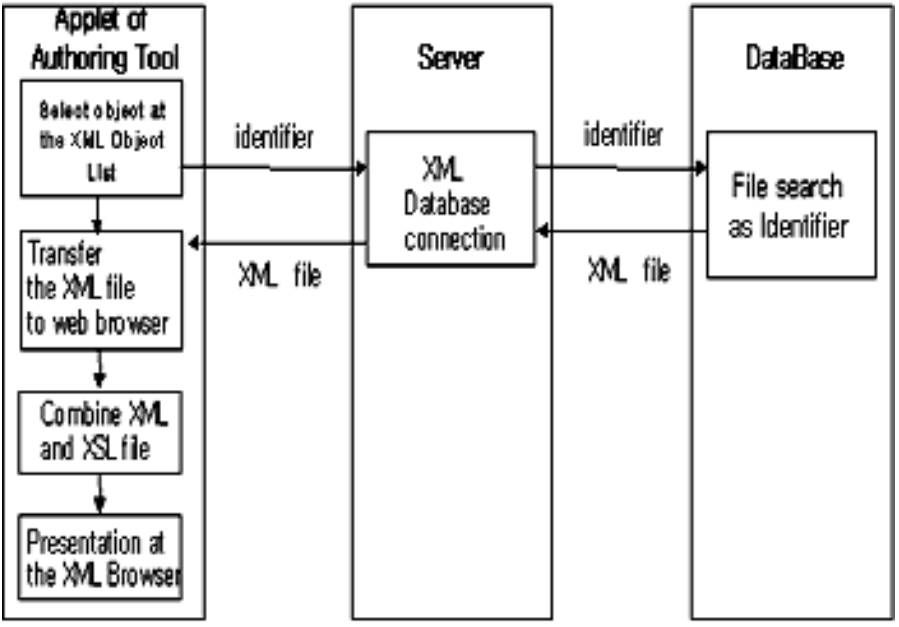


Fig. 3. Flowchart for the execution of XML Data

When the development applet is executed, it sends a request to the server to retrieve the product categories and product names in each product category. The server gets these data from the database and it sends them to the development applet, then the development applet shows these data to the developer. If the developer chooses one of the products, the development applet sends the product identity of the product chosen to the server and requests its information and its Virtual Object file. The server gets the product information and the Virtual Object file that corresponds to the product identity received from the database. Afterwards it sends them to the development applet. Next, upon receiving these data the development applet shows the product information to the developer and transfers the Virtual Object file to the Virtual Object translator in order to make the Virtual Object file into a Virtual Object. The Virtual Object translator translates the Virtual Object file into Java class, called a Classnode, corresponding to each Virtual Object node in the Virtual Object file. Then as these Java classes are loaded into memory it makes a Virtual Object from the Virtual Object file.

The developer can manipulate these Virtual Objects. If the developer wants to create an e-Catalog, he/she enters catalog name and selects save. The Virtual Object code extractor calls a method in Classnode to extract the code loaded in memory and then merges the extracted code into one Virtual Object file. It is then sent to the server to be stored on the database.

Virtual Object translator. The Virtual Object translator is included in both the development applet and the customer applet. First, in order to use Virtual Object file in Java, it creates a Virtual Object instance and It parses the Virtual Object file into tokens. Then it distinguishes whether the token is a node statement or a product identity. If the token is a node statement, the Virtual Object translator creates a Classnode instance and calls a translating function corresponding to the node type. Then the function fills up variables with values from the parsed nodes. If the token is a product identity, the Virtual Object translator inserts the product identity into the product ID variable in a Virtual Object instance. The Virtual Object translated is inserted into the Virtual Object scene then it can be moved or deleted. This is presented in the flowchart in Figure 2.

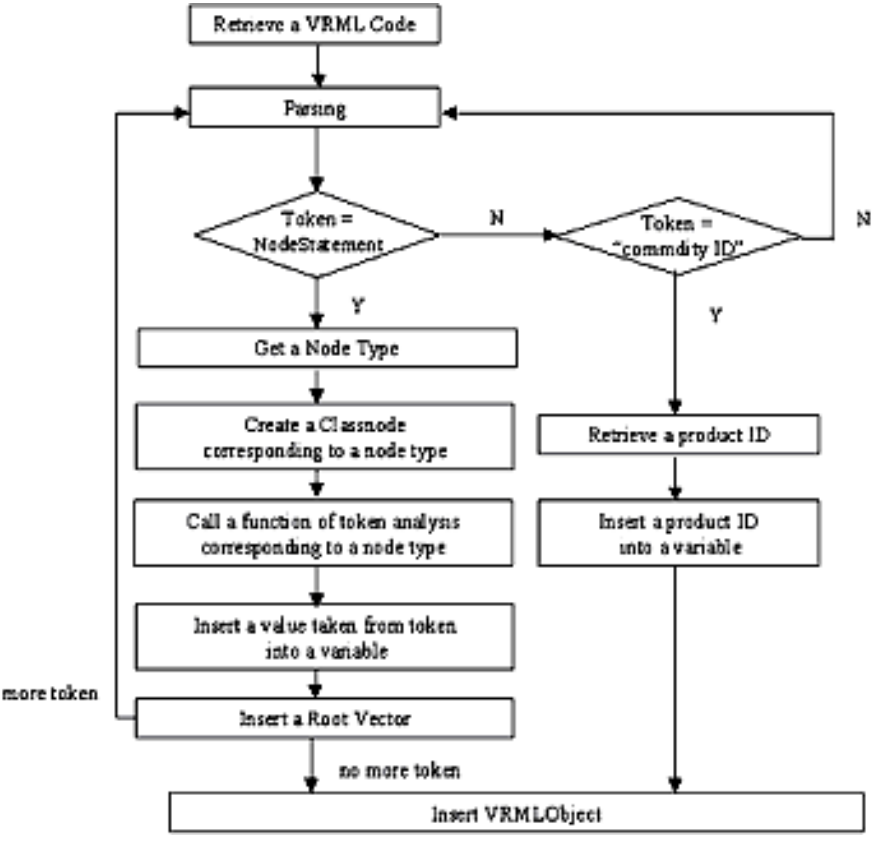


Fig. 4. The Flowchart of the Virtual Object translating

Virtual Object code extractor. The Virtual Object code extractor makes Virtual Object codes from Virtual Objects loaded in memory. In order to create an e-Catalog, the developer selects save on the user interface. Then the Virtual Object code extractor gets all the Virtual Objects through typecasting as ObjectInfo class type, which is the parent class of Virtual Object and then it calls extractor method declared as an abstract method of Java. Therefore each extractor method of Virtual Object gets its field names and values from Classnode in Virtual Object instance and prints them as a String object. Finally it merges the extracted code into one Virtual Object file.

4 Implementation

This tool is implemented under Windows 2000 Advanced Server, JDK 1.3.0, Oracle 8i database and Blaxxun 4.4 VRML browser. We show the implementation of tool through an example. First, we get a Virtual Object file and enter product information and a product identity. Results are shown in Figure 3. Then we select save to insert them into the database.

And we execute rmiregistry of Java then run the server. Second, we run the development applet on the Web. We choose a product on the item list and select load to get a product and then it can be manipulated. Results are shown in Figure 4.

Afterwards we retrieve products through this way and manipulate them, then we enter a catalog name and select save to create an e-Catalog. Afterwards we run customer applet on the Web. Next we choose a catalog name that we made on the development applet in the catalog selection box. Then we choose a product. Results are shown as the product information in the information text field in Figure 5.

5 Evaluation

This chapter is compared a mechanism for interoperating with database of the existing Virtual object and the suggested method in this paper. The next table is that.

Table 1. Comparison of the two method

Items	This Method	Embedded SQL	ServerRedirect SQL
Database Interconn -ection Level	Application	Script	Script
Interoperating with the other application	Interoperating with the java application	Script language	Script language
Support of the other data format	Possible	Text only	Text only

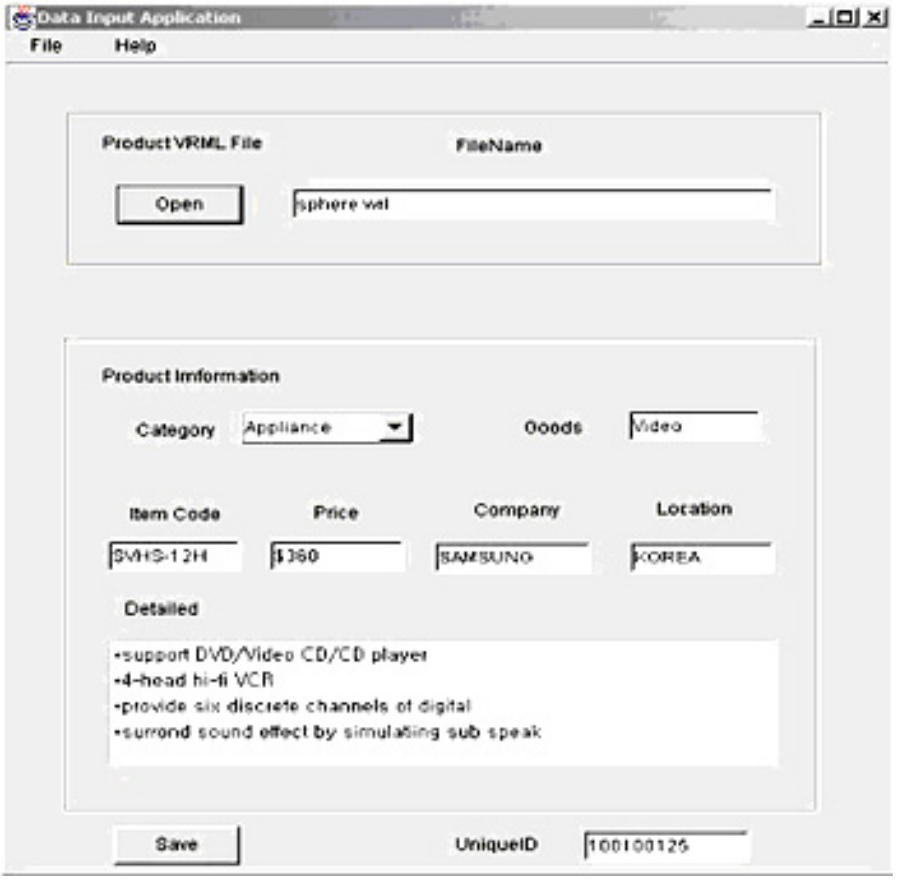


Fig. 5. Entering product information in Data Input Application

The existing method must be updated directly at the source level in the virtual object reconstruction but this method can be controlled at the high level. In the conclusion, we are measured execution time of upper methods. The values of result are average of the execution time at the query of 100 time. This result is Table 2.

Table 2. Execution Time

Items	This Method	Embedded SQL	ServerRedirect SQL
Database Interconnection	160ms	182ms	183ms

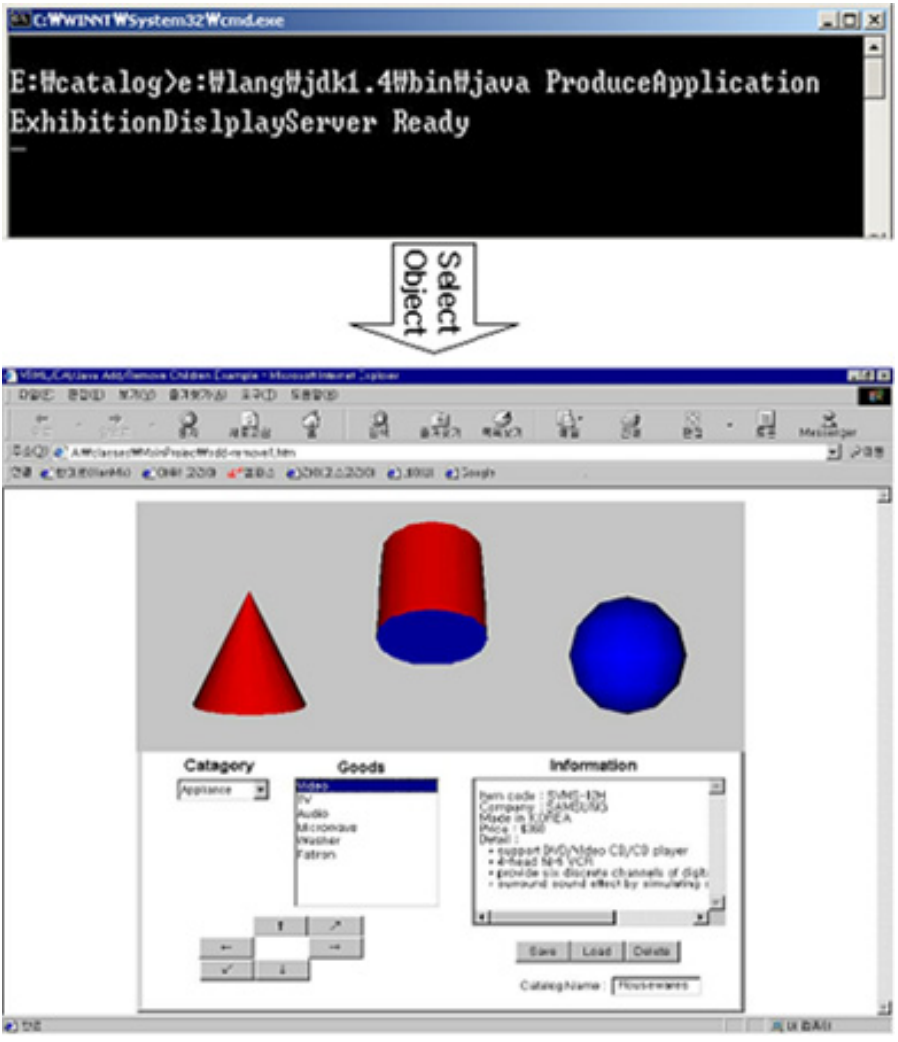


Fig. 6. Manipulating products in the development applet

The difference of the execution time presented to the upper result is a few but the system is bigger, the effects will be better.

6 Conclusion and Future Work

There are many authoring tools that can create multimedia-contents using Virtual Object, but these authoring tools cannot be convenient to user of the authoring tool. That is, existing Virtual Object authoring tools isn't convenient to user of the authoring tool. In this paper we propose an authoring tool that

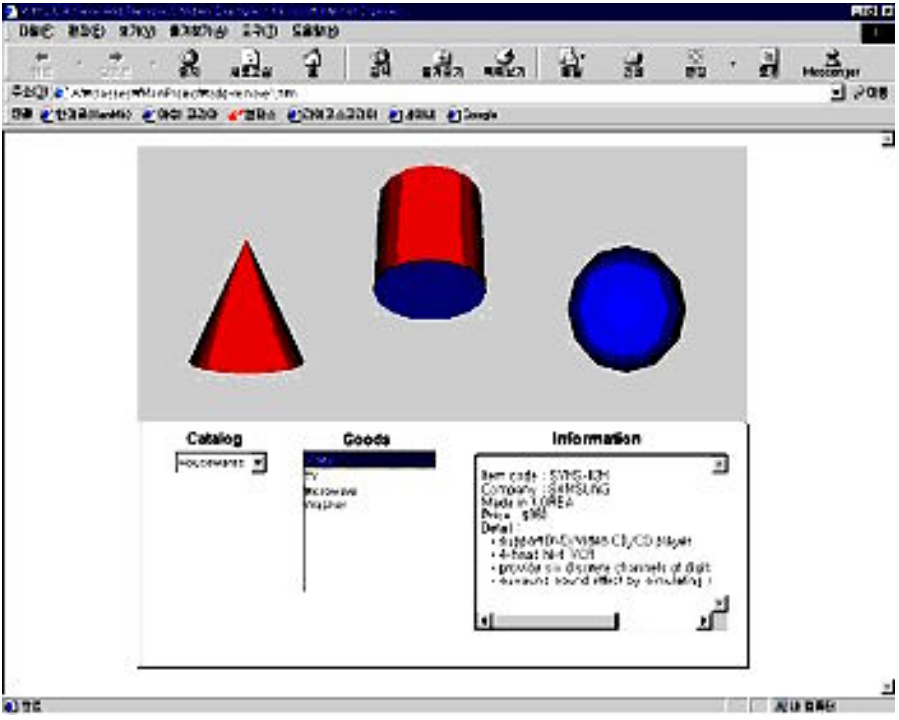


Fig. 7. Choosing a product on the customer applet

integrates Virtual Object and database so that a 3D e-Catalog based on Virtual Object connection to DB can be made. Therefore developers are offered the ability to manipulate Virtual Object on a high-level.

In this design for the upper advantage we separated Virtual Object control logic and database control logic as we implemented the development applet using EAI and the Client/Server Model. With this we have solved problems in the method used when connecting to the database in existing Virtual Object authoring tools. The Virtual Object translator implemented dynamically controls Virtual Objects and the Virtual Object code extractor implemented makes various Virtual Objects translated on Virtual Object translator to an e-Catalog Virtual Object file - One of the authoring tools in this paper applied with the e-Catalog Virtual Object file for request of electronic-commerce now.

In further studies, we will investigate converting product information into XML format to support ebXML, standard of e-Catalog and to change the appearance of products for JAVA 3D to handle products on the Web.

References

1. The Virtual Reality Modeling Language (International Standard ISO/IEC 14772-1:1997), Web3d consortium, (1997) http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm
2. Cosmo Worlds, Cosmo Software. http://www.cai.com/cosmo
3. Internet Space Builder, ParallelGraphics. http://www.parallelgraphics.com
4. Spazz3D, Virtock Technology, Inc. http://www.spazz3d.com
5. Web3d consortium, (1998) http://www.web3d.org/Recommended/vrml-sql
6. External Authoring Interface Group, (1999) http://www.web3d.org/WorkingGroups/vrml-eai
7. A. Heinonen, S. Pulkkinen, I. Rakkolainen: An Information Database for VRML Cities. Proceedings of the International Conference on Information Visualization. (2000) 469–473
8. Michel Buffa, Jean-Claude Lafon: 3D Virtual Warehouse on the WEB. Proceedings of the International Conference on Information Visualization. (2000) 479–484
9. Kaoru Sugita, Akihiro Miyakawa, Yukiharu Kohsaka, Koji Hashimoto, Yoshitaka Shibata: Traditional Japanese Crafting Presentation System Based on VR and Kansei Information Processing Techniques. ICPADS'00 Workshops. (2000) 73–77
10. Kris A. Jamsa, Phil Schmauder, Nelson Yee: VRML Programmer's Library. JAMSA Press. (1997)

Language Standardization for the Semantic Web: The Long Way from OIL to OWL

D. Fensel

Vrije Universiteit Amsterdam (VU)
Faculty of Sciences, Division of Mathematics and Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
Fax and Answering machine: +31-(0)84-872 27 22, Mobil phone: +31-(0)6-51850619
dieter@cs.vu.nl

Abstract. The major objective for the development of OIL was to develop an international standard for web-based representation of ontological information. This papers reports some lessons learnt in this process.

1 Introduction

"The Semantic Web is very exciting, and now just starting off in the same grassroots mode as the Web did 10 years ago... In 10 years it will in turn have revolutionized the way we do business, collaborate and learn."
Tim Berners-Lee, CNET.com interview, 2001-12-12

The World Wide Web (WWW) has drastically changed the availability of electronically accessible information. The WWW currently contains some 3 billion static documents, which are accessed by over 300 million users internationally. At the same time, this enormous amount of data has made it increasingly difficult to find, access, present, and maintain the information required by a wide variety of users. This is because information content is presented primarily in natural language. Thus, a wide gap has emerged between the information available for tools aimed at addressing the problems above and the information maintained in human-readable form.

In response to this problem, many new research initiatives and commercial enterprises have been set up to enrich available information with machine-processable semantics. Such support is essential for "bringing the web to its full potential". Tim Berners-Lee, Director of the World Wide Web Consortium, referred to the future of the current WWW as the "semantic web" – an extended web of machine-readable information and automated services that extend far beyond current capabilities [2,15]. The explicit representation of the semantics underlying data, programs, pages, and other web resources, will enable a knowledgebased web that provides a qualitatively new level of service. Automated services will improve in their capacity to assist humans in achieving their goals by "understanding" more of the content on the web, and thus providing more accurate filtering, categorization, and searches of information sources. This process will ultimately lead to an extremely knowledgeable system that features

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 215–227, 2002.

various specialized reasoning services. These services will support us in nearly all aspects of our daily life – making access to information as pervasive, and necessary, as access to electricity is today.

Ontoknowledge[1] [10,14]) is an European IST project that develop such semantic web technology. Aim is the support of knowledge management for weakly structured, large, heterogeneous, and distributed information sources. The competitiveness of many companies depends heavily on how they exploit their corporate knowledge and memory. Most information in modern electronic media is mixed media and rather weakly structured. This is not only true of the Internet but also of large company intranets. Finding and maintaining information is a tricky problem in weakly structured representation media. Increasingly, companies have realized that their intranets are valuable repositories of corporate knowledge. But as volumes of information continue to increase rapidly, the task of turning them into useful knowledge has become a major problem. Therefore, Ontoknowledge builds an ontology-based tool environment to speed up knowledge management, dealing with large numbers of heterogeneous, distributed, and semi-structured documents typically found in large company intranets and the World Wide Web. The project's target results are: (1) a toolset for semantic information processing and user access; (2) OIL, an ontology-based inference layer on top of the World Wide Web; (3) an associated methodology and validation by industrial case studies.

Effective and efficient work with ontologies must be supported by advanced tools that allow users to put this technology to its fullest use. In particular, we need an advanced ontology language to express and represent ontologies. The major objective for the development of this ontology language was not to create yet another knowledge representation language that will be used internally in a small project. We rather decided to aim for an international standard. Clearly this would help to disseminate our project results and it would improve the overall visibility of Ontoknowledge. This paper is about how this approach worked out.

The content of the paper is organized as follows. In Section 2, we describe the initial results of our effort, the language OIL. Section 3 describes the intermediate result which was called DAML+OIL, and Section 4 provides a look at ongoing standardization effort at the W3C[2], the recommendation committee of the World Wide Web. Section 5 reports about the internal use of OIL, i.e., the use of OIL in Ontoknowledge. Finally, Section 6 provides an outlook on additional areas that will be soon covered by standardization effort related to the semantic web.

2 Initial Results: OIL

We produced a detailed survey that summarized the state of the art on Ontology languages, that summarized the state of the art on related web standards, and

[1] `http://www.ontoknowledge.org/`
[2] `http://www.w3c.org/`

that came up with a proposal for an Ontology Web Language based on four principles [9]:[3]

OIL, a modeling language based on frames: The central modeling primitives of predicate logic are predicates. Frame-based and object-oriented approaches take a different point of view. Their central modeling primitives are classes (i.e., frames) with certain properties called attributes. These attributes do not have a global scope but are only applicable to the classes they are defined for (they are typed) and the "same" attribute (i.e., the same attribute name) may be associated with different range and value restrictions when defined for different classes. A frame provide a certain context for modeling one aspect of a domain. Many other additional refinements of these modeling constructs have been developed and have led to the incredible success of this modeling paradigm. Many frame-based systems and languages have been developed and, renamed as object-orientation they have conquered the software engineering community. Therefore, OIL incorporates the essential modeling primitives of frame-based systems into its language.

OIL, a language where everybody can ask for its formal semantics based on experience in Description Logics (DL): DLs describe knowledge in terms of concepts and role restrictions that are used to automatically derive classification taxonomies. The main effort of research in knowledge representation is in providing theories and systems for expressing structured knowledge and for accessing and reasoning with it in a principled way. In spite of the discouraging theoretical complexity of their results, there are now efficient implementations for DL languages, see for example DLP and the FaCT system [17]. OIL inherits from Description Logic its formal semantics and the efficient reasoning support developed for these languages. In OIL, subsumption is decidable and with FaCT we can provide an efficient reasoner for this.

OIL, a language with a web compatible syntax based on XML and RDF: Modeling primitives and their semantics are one aspect of an Ontology language. Second, we have to decide about its syntax. Given the current dominance and importance of the WWW, a syntax of an ontology exchange language must be formulated using existing web standards for information representation. First, OIL has a well-defined syntax in XML based on a DTD and a XML schema definition. Second, OIL is defined as an extension of the Resource Description Framework RDF and RDFS.[4]

OIL, a solution for people who need a bike and people who like a Mercedes: It is unlikely that a single ontology language can fulfill all the needs of the large range of users and applications of the Semantic Web. We have therefore organized OIL as a series of ever increasing layers of sub languages. Each additional layer adds functionality and complexity to the previous layer. This is done such that agents (humans or machines) who can only process a lower layer can still partially understand ontologies that are expressed in any of the higher layers. The layered architecture of OIL has three main advantages:

[3] A fifth principle was to come up with a good name.

[4] `http://www.w3c.org/RDF/`

First, an application is not forced to work with a language that offers significant more expressiveness and complexity that it actually needs. Second, application that can only process a lower level of complexity are still able to catch same of the aspects of an ontology. Third, an application that is aware of a higher level of complexity can still also understand ontologies expressed in a simpler ontology language. A first and very important application of this principle is the relation between OIL and RDF Schema. As shown in the Figure 1, **Core OIL** coincides largely with RDF Schema (with the exception of the reification features of RDF Schema). This means that even simple RDF Schema agents are able to process the OIL ontologies, and pick up as much of their meaning as possible with their limited capabilities. **Standard OIL** is a language intended to capture the necessary mainstream modelling primitives that both provide adequate expressive power and are well understood thereby allowing the semantics to be precisely specified and complete inference to be viable. **Instance OIL** includes a thorough individual integration. **Heavy OIL** may include additional representational (and reasoning) capabilities.

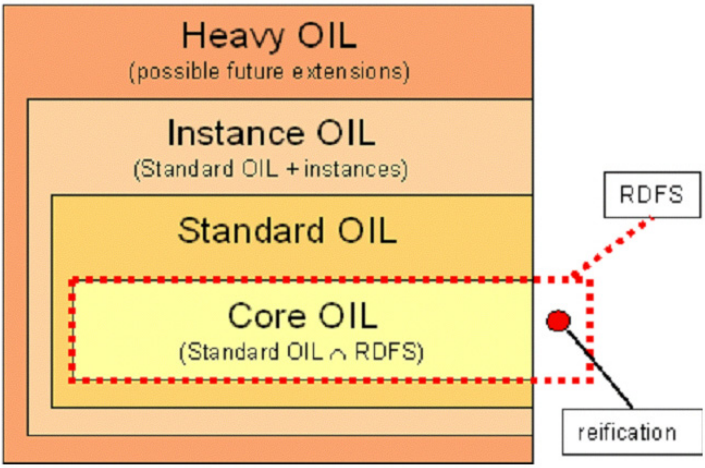


Fig. 1. The Onion model of OIL.

Accompanied with the right spirit and coming up in the right moment we could quickly acquire a large group of people that joined our vision and helped us in building up a widely visible proposal for an Ontology Web Language. Most of these results, cooperation partners, and events are mentioned in [12]. A highly visible web site was installed at `http://www.ontoknowledge.org/oil` to collect OIL resources. A large number of publications on OIL have been produced from which we just mention some of them: [5,6,8,11,19,20]. Meanwhile, many Ontology tools developed inside and outside the Ontoknowledge project support OIL.

Figure 2 provides a very simple example of an OIL ontology. It only illustrates the most basic constructs of OIL. This defines a number of classes and organizes them in a classhierarchy (e.g. HPProduct is a subclass of Product). Various properties ("slots") are defined, together with the classes to which they apply (e.g. a Price is a property of any Product, but a PrintingResolution can only be stated for a Printer (an indirect subclass of Product). For certain classes, these properties have restricted values (e.g. the Price of any HPLaserJet1100se is restricted to be $479). In OIL, classes can also be combined using logical expressions, for example: an HPPrinter is both an HPProduct and a Printer (and consequently inherits the properties from both these classes).

```
class-def Product
slot-def Price
  domain Product
slot-def ManufacturedBy
  domain Product
class-def
PrintingAndDigitalImagingProduct
  subclass-of Product
class-def HPProduct
  subclass-of Product
  slot-constraint ManufacturedBy
    has-value "Hewlett Packard"
class-def Printer
  subclass-of
  PrintingAndDigitalImagingProduct
slot-def PrinterTechnology
  domain Printer
slot-def Printing Speed
  domain Printer
slot-def PrintingResolution
  domain Printer
class-def PrinterForPersonalUse
  subclass-of Printer
class-def HPPrinter
  subclass-of HPProduct and Printer
class-def LaserJetPrinter
  subclass-of Printer
  slot-constraint PrintingTechnology
    has-value "Laser Jet"
class-def HPLaserJetPrinter
  subclass-of LaserJetPrinter
    and HPProduct
class-def HPLaserJet1100Series
  subclass-of HPLaserJetPrinter
    and PrinterForPersonalUse
  slot-constraint PrintingSpeed
    has-value "8 ppm"
  slot-constraint PrintingResolution
    has-value "600 dpi"
class-def HPLaserJet1100se
  subclass-of HPLaserJet1100Series
  slot-constraint Price
    has-value "$479"
class-def HPLaserJet1100xi
  subclass-of HPLaserJet1100Series
  slot-constraint Price
    has-value "$399"
```

Fig. 2. A simple OIL Ontology.

3 Intermediate Results: DAML+OIL

When a small European project touches the orbit of a large US project cluster like DAML[5] it is time for co-operation. DAML is a significantly funded project that puts 80 Million Dollar to develop semantic web technology. Many of the core

[5] http.//www.daml.org/

members of OIL joined the DAML-ONT working group and in weekly telephone conferences a new language called DAML+OIL was born. A new body the *Joint EU/US ad hoc Agent Markup Language Committee*[6] was installed and has now been running for more than a year. In this process, we had to throw our OIL principles number 1, 4, and 5 over board, however, we came up with a truly international standard backed up by the US defence department. Most DAML+OIL results can be found at `http://www.daml.org/2000/12/daml+oil-index.htm` and `http://www.daml.org/2001/03/reference.html`. Basically, DAML+OIL is a (rather expressive) description logic with web syntax. Neither does it provide layered sub languages with different complexity nor language primitives that are defined around a modeling paradigm.

4 Ongoing Final Results: OWL

Meanwhile the W3C[7], the recommendation committee of the World Wide Web, has set up a *Semantic Web Activity*[8] and as part of it a *Web-Ontology (WebOnt) Working Group*[9]. Working in this group requires weekly phone conferences, regular face-to-face meetings and dealing with high email traffic[10]. Final results are expected for September 2002 and it is just too early to predict them. Working since late 2001 this working group deals with issues such as:

- What is the right name for the Ontology Web Language (OWL)?[11]
- What are the significant use cases that justify the definition of certain language primitives?
- What is the right layering approach for the Semantic Web (see Figure 3). Should OWL be based on RDF and does it matter that their model theories entail different conclusions?

OIL was defined as an extension of RDF-Schema which means that every RDF-Schema ontology is a valid ontology in the new language (i.e., an OIL processor will also understand RDF Schema). However, the other direction is also available: defining an OIL extension as closely as possible to RDF Schema allows maximal reuse of existing RDF Schema-based applications and tools. However, since the ontology language usually contains new aspects (and therefore new vocabulary, which an RDF Schema processor does not know), 100% compatibility is not possible. Let us examine an example. The following OIL expression defines herbivore as a class, which is a subclass of animal and disjunct to all carnivores.

[6] `http://lists.w3.org/Archives/Public/www-rdf-logic/2000Nov/0093.html`
[7] `http://www.w3c.org/`
[8] `http://www.w3.org/2001/sw/Activity`
[9] `http://www.w3.org/2001/sw/WebOnt/`
[10] `http://lists.w3.org/Archives/Public/www-webont-wg/`
[11] `http://www.cs.vu.nl/~frankh/spool/names.html`

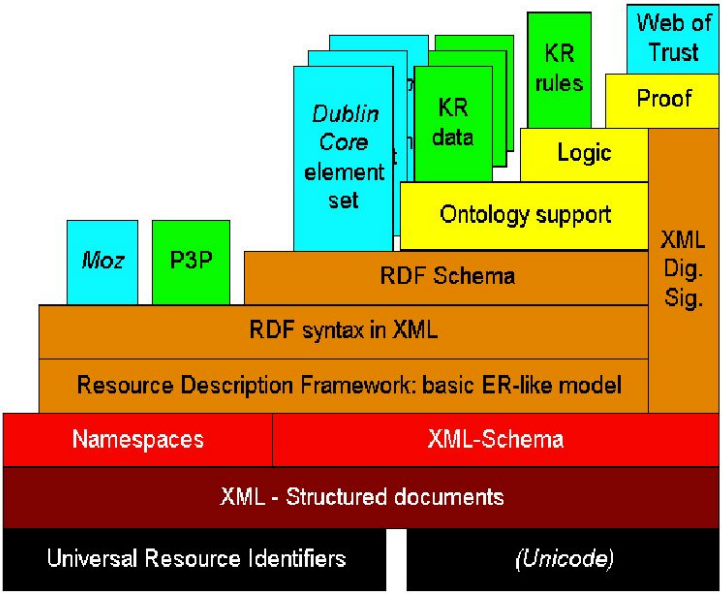


Fig. 3. The layer language model for the semantic web.

```
<rdfs:Class rdf:ID="herbivore">
  <rdf:type
    rdf:resource="http://www.ontoknowledge.org/oil/RDFS-schema/
    #DefinedClass"/>
  <rdfs:subClassOf rdf:resource="#animal"/>
  <rdfs:subClassOf>
    <oil:NOT>
      <oil:hasOperand rdf:resource="#carnivore"/>
    </oil:NOT>
  </rdfs:subClassOf>
</rdfs:Class>
```

An application limited to pure RDFS is still able to capture some aspects of this definition.

```
<rdfs:Class rdf:ID="herbivore">
  <rdfs:subClassOf rdf:resource="#animal"/>
  <rdfs:subClassOf>
  ...
  </rdfs:subClassOf>
</rdfs:Class>
```

It sees that herbivore is a subclass of animal and a subclass of a second class, which it cannot understand properly. This seems to be a useful way to preserve complicated semantics for simpler applications.

Recently, a interesting feature of this approach become visible. In general one would assume that an OIL/OWL agent can draw more conclusions than an RDFS aware agent. This I call the *model theoretic interpretation* of the semantic web language tower. However, based on the RDF model theory [16] this turned out to be not true. Because RDF "sees" the syntactical definition of an ontology it can draw conclusions that OWL, which is situated at a logical level, cannot. That is, not every model for an OWL ontology is also a model for the underlying RDF representation.[12] This is caused by the fact, that RDF treats a statement like `a disjoint b` as a syntactical expression that it represents as a relation between a and b in its model theory whereas OWL defines in its model theory that the concepts a and b are disjoint. Therefore, defining the model theory of OWL as an extension of the model theory of RDF **and** representing OWL constructs syntactically in RDF leads to paradoxical situations, i.e., ill-defined model theories for OWL. Currently, there are two positions on this problem:

Semantical enrichment: The model theory of OWL is defined in extension of the model theory of the underlying language RDF. This requires that many syntactical constructs of OWL are not defined in RDF but in XML, for example. This prevents paradoxical situations but breaks with the idea of syntactical enrichments of the various layers.

Syntactical Enrichment: One can see entailment defined in RDF as a means to reason about the syntax of a language or Ontology (with RDF reasoning you can make a distinction whether a disjoint statement is defined explicitly or whether it is a logical conclusion of other logical statements) and entailment defined in OWL as a means to reason about the semantics of an Ontology. Therefore, RDF should not viewed as a basic Ontology language but as a syntactical mechanism to define Ontology Languages in it. From the standpoint of an Ontology language, RDF has many strange features. On the other hand, they may make sense if you want to use RDF as a mechanism to define Ontology languages in it.

The latter approach is quite interesting because it refers to the problem on how to represent and infer modeling information in a logical framework. Lets take a simple example. A statement like

$$a \wedge b \tag{1}$$

should be treated completely equal to a statement

$$b \wedge a \tag{2}$$

at a logical level. However, at a modeling level you may want to know whether a person wrote (1) or (2) because it may reflect the fact that b is more "important" to him. A more real-life example is whether a person define

[12] `http://lists.w3.org/Archives/Public/www-webont-wg/2001Dec/0116.html`

- a relation r as an attribute of a class r or
- as a global property r with c as its domain.

Logically they are the same but from a modeling point of view they are quite different. By having two types of entailments we can capture this without running into any problems. With syntactical RDF reasoning we can ask for different syntactical styles of an Ontology and with semantic OWL reasoning we infer logical consequences of an Ontology. This view point would also allow us to deal with different modeling paradigms people are asking for. We could just define a frame syntax for OWL in RDF [1] making sure that it behaves the same as the non-frame version at the logical level but behaves different at the syntactical level, i.e., in the frame version you could ask whether something is explicitly defined as an attribute or as a property. In a nutshell, it is a very interesting feature having a reasoning level that reasons "non"-logically but syntactically over an OWL Ontology. It is the layer that allows us to capture, infer, and query modeling information. Therefore, this type of information no longer needs to be messed up with the logical layer, where indeed you do not want to care about the different syntax of logically equivalent statements.

In any case, figuring out the proper principles for building up the semantic web language tower requires more work and recalls earlier approaches in *knowledge representation* [3,4] and *knowledge modeling* [23].

5 Why Not Eat Your Own Dogs' Food

Compared with the external use and adaptation of OIL, the internal use of OIL within Ontoknowledge in case studies as well as in the developed tools[13] is rather disappointing. Ontoknowledge is strongly driven by several case studies such as finding relevant passages in a very large document about the International Accounting Standard (IAS) on the extranet (over 1000 pages); a skills management application that uses manually constructed ontologies about skills, job functions, and education; exchanging knowledge in a virtual organization; and providing knowledge management support for a call centre.[14] None of the case studies really employ OIL. They all refer to RDF Schema or a subset of it which we called **Core OIL** to mask our "failure". What are the reasons for this:

- Some case studies began even before the first version of OIL was available.
- OIL lacks any tutorial support, any customized tool support, and any real practical experience. From a user point of view this makes it nearly unusable for the moment.

[13] The developed tools include (1) RDFferret which combines full text searching with RDF querying; (2) OntoShare enables the storage of best practice information in an ontology; (3) Spectacle organizes the presentation of information; (4) OntoEdit makes it possible to inspect, browse, codify and modify ontologies; (5) Sesame is a system that allows persistent storage of RDF data and schema information and subsequent online querying of that information; and information extraction and ontology generation is performed by means of the CORPORUM toolset.

[14] `http://www.ontoknowledge.org/del.shtml`

– OIL or OWL are ongoing standardization efforts including high change rates. No user nor tool developer can trust the current language version as being final and not just an intermediate step. Using OIL is of high risk, both for uses in case studies as well as for tool developers.

The lack of actual use of OIL may also be an indication for three different problems related to OIL:

OIL is too little expressive. Many things cannot be expressed in it but could be easily expressed in a rule-based language like F-logic [18] oriented on reasoning over instances. The initial approaches to the semantic web were oriented around this paradigm [22,7].

OIL is too expressive. OIL is logic based and OIL is description logic based. Many people find it difficult or not worth while to express themselves within logic. For example, non of the standard ontologies in electronic commerce or widely used ontologies such as Wordnet[15] make use of any axiomatic statements. Mostly they are simple taxonomies enriched by attributes in the best case. Description logic adds a specific feature: Concept hierarchies do not need to be defined explicitly by can be defined implicitly by complex definitions of classes and properties. Many people may find it easier to directly define is-a relationships instead of enforcing them by complex and well-thought axiomatic definition of classes and properties.

Ontologies should not be based on formal logic. People with a database background wonder in general whether axioms, i.e., complex logical statements should be part of an ontology. They tend to be application specific and very difficult to exchange and reuse in a different context. Spoken frankly, most of our experience conform with this statement. Still we may hope to make the world a better place in the future enabling more than just the exchange of concept taxonomies and some attributes.

In any case, it is too early to take a final conclusion from our exercise. We hope that we will get more insights from the use cases collected by the *Web-Ontology (WebOnt) Working Group*[16]. Also the *SIG on Ontology Language Standards*[17] of *Ontoweb*[18] is supposed to bring further insights here.

6 Outlook

It is much too early to give an answer to this question. Anyway, this mixture of model theory on the one side and sociology on the other side is quite interesting and unique.[19] Many new interesting issues are completely uncovered by the work described here:

[15] `http://www.cogsci.princeton.edu/ wn/`
[16] `http://www.w3.org/2001/sw/WebOnt/`
[17] `http://www.cs.man.ac.uk/ horrocks/OntoWeb/SIG/`
[18] `http://www.ontoweb.org/`
[19] Not really when reading [21].

The lack of rules: We already mentioned the fact that early approaches for the semantic web were based on rule-based approaches. Also their is a clear consensus that the current Ontology Web Language needs an extension in this direction. The *Rule Markup Language Initiative*[20] may help to make a step into this direction.

The language tower of the semantic web: Giving a real semantics to the semantic web language tower as sketched by Tim Berners-Lee in Figure 3 requires much more work. Currently many layering ideas oriented to syntactical and semantical extensions compete which each other.[21] Working out the proper relationship will be much more challenging than just developing one layer for it.

Web Services: The easy information access based on the success of the web has made it increasingly difficult to find, present, and maintain the information required by a wide variety of users. In response to this problem, many new research initiatives and commercial enterprises have been set up to enrich available information with machine-understandable semantics. This **semantic web** will provide intelligent access to heterogeneous, distributed information, enabling software products to mediate between user needs and the information sources available. **Web Services** tackle with an orthogonal limitation of the current web. It is mainly a collection of information but does not yet provide support in processing this information, i.e., in using the computer as a computational device.

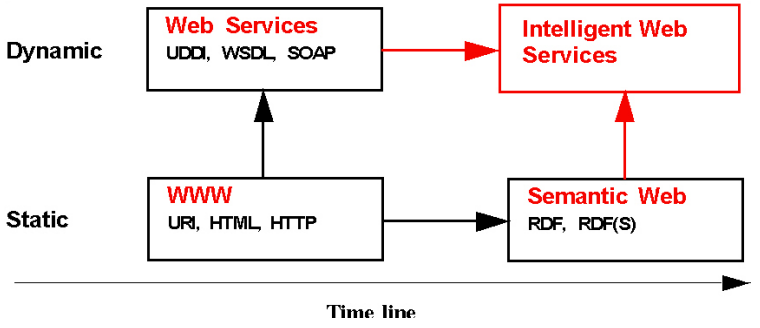


Fig. 4. The development process of the web.

Recent efforts around UDDI[22], WSDL[23], and SOAP[24] try to lift the web to a new level of service. Software programs can be accessed and executed via the web. However, all these service descriptions are based on semi-formal natural language

[20] http://www.dfki.uni-kl.de/ruleml/
[21] see http://lists.w3.org/Archives/Public/www-webont-wg/
[22] http://www.uddi.org/
[23] http://www.wsdl.org/
[24] http://www.soap.org/

descriptions. Therefore, the human programmer need be kept in the loop and scalability as well as economy of web services are limited. Bringing them to their full potential requires their combination with semantic web technology. It will provide mechanization in service identification, configuration, comparison, and combination. **Semantic Web enabled Web Services** have the potential to change our life in a much higher degree as the current web already did (Figure 4). We are currently in the process of extending UPML [13] to a full-fledged **Web Service Modeling Language** (**WSML**).

Acknowledgement. Too many people were involved in working out OIL to mention all of them. Therefore, I just would like to mention Ian Horrocks and Frank van Harmelen. In any case, James Hendler devotes many thanks for initiating the field and strongly promoting it with the DAML project cluster.

References

1. S. Bechhofer, C. Goble, and I. Horrocks: DAML+OIL is not enough. SWWS-1, Semantic Web working symposium, Stanford (CA), July 29th-August 1st, 2001.
2. T. Berners-Lee, J. Handler, and O. Lassila: The Semantic Web, Scientific American, May 2001.
3. R. J. Brachman: What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks, IEEE Computer, 16(10), October 1983.
4. R. J. Brachman: The Myth of the One True Logic, Computer Intelligence, 3(3), August 1987.
5. J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks: Enabling knowledge representation on the Web by extending RDF Schema, to appear in Electronic Transactions on Artificial Intelligence (ETAI).
6. S. Decker, F. van Harmelen, J. Broekstra, M. Erdmann, D. Fensel, I. Horrocks, M. Klein, and S. Melnik: The Semantic Web – on the respective Roles of XML and RDF, IEEE Internet Computing, September/October 2000.
7. D. Fensel, S. Decker, M. Erdmann, and R. Studer: Ontobroker: The Very High Idea. In Proceedings of the 11th International Flairs Conference (FLAIRS-98), Sanibal Island, Florida, USA, 131–135, May 1998.
8. D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein: OIL in a nutshell. In Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference (EKAW-2000), R. Dieng et al. (eds.), Lecture Notes in Artificial Intelligence, LNAI 1937, Springer-Verlag, October 2000.
9. D. Fensel, I. Horrocks, F. van Harmelen, J. Broekstra, M. Crubezy, S. Decker, Y. Ding, M.Erdmann, C. Goble, M. Klein, B. Omelayenko, S. Staab, H. Stuckenschmidt, and R. Studer: Ontology Language Version 1, Deliverable No 1 of the IST project Ontoknowledge, 2000, see `http://www.ontoknowledge.org/del.shtml`
10. D. Fensel, F. van Harmelen, H. Akkermans, M. Klein, J. Broekstra, C. Fluyt, J. van der Meer, H.-P. Schnurr, R. Studer, J. Davies, J. Hughes, U. Krohn, R. Engels, B. Bremdahl, F. Ygge, U. Reimer, and I. Horrocks: OnToKnowledge: Ontology-based Tools for Knowledge Management. In Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference, Madrid, Spain, October 2000.

11. D. Fensel, I. Horrocks, F. van Harmelen, D. McGuinness, and P. F. Patel- Schneider: OIL: Ontology Infrastructure to Enable the Semantic Web, IEEE Intelligent System, 16(2), 2001.
12. D. Fensel, F. van Harmelen, and I. Horrocks: OIL: A Standard Proposal for the Semantic Web, Deliverable No 0 of the IST project Ontoknowledge, 2001, see `http://www.ontoknowledge.org/del.shtml`
13. D. Fensel, E. Motta, F. van Harmelen, V. R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga: The Unified Problem-solving Method Development Language UPML, to appear in Knowledge and Information Systems (KAIS): An International Journal.
14. D. Fensel, F. van Harmelen, Y. Ding, M. Klein, H. Akkermans, J. Broekstra, A. Kampman, J. van der Meer, Y. Sure, R. Studer, U. Krohn, J. Davies, R. Engels, V. Iosif, A. Kiryakov, T. Lau, and U. Reimer: On-To-Knowledge: Semantic Web Enabled Knowledge Management, submitted to IEEE Computer.
15. D. Fensel and M. Musen: Special Issue on Semantic Web Technology, IEEE Intelligent Systems (IEEE IS), 16(2), 2001.
16. P. Hayes: RDF Model Theory W3C Working Draft, 14 December 2001. `http://www.coginst.uwf.edu/users/phayes/w3-rdf-mt-current-draft.html`
17. I. Horrocks: Using an expressive description logic: Fact or fiction? In Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2–5, 1998.
18. M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, Journal of the ACM, 42, 1995.
19. M. Klein, J. Broekstra, D. Fensel, F. van Harmelen, and I. Horrocks: Ontologies and schema languages on the Web. In D. Fensel et al. (eds.), Semantic Web Technology, MIT Press, Boston, to appear 2002.
20. M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks: The relation between ontologies and XML schemas, to appear in Electronic Transactions on Artificial Intelligence (ETAI).
21. B. Latour: One More Turn after the Social Turn: Easing Science Studies into the Non-Modern World. In Ernan McMullin (ed.), The Social Dimensions of Science, Notre Dame University Press: Notre Dame, pp. 272–292, 1992. See `http://www.ensmp.fr/~latour/`
22. S. Luke, L. Spector, D. Rager, and J. Hendler: Ontology-based Web Agents. In Proceedings of First International Conference on Autonomous Agents 1997, AA-97.
23. A. Newell: The Knowledge Level, Artificial Intelligence, vol 18, 1982.

Noema: A Metalanguage for Scripting Versionable Hypertexts

Ioannis T. Kassios and M.C. Schraefel

Department of Computer Science, University of Toronto
Toronto ON M5S 3G4 Canada
{ykass, mc}@cs.toronto.edu

Abstract. In this paper we present Noema, a new intensional hyper-text metalanguage based on both intensional HTML and XML. The design of Noema aims at the improvement of the expressiveness of intensional HTML and introduces principles such as *intensional XML-like entities*, *versioning / hypertext unification* and *implicitly created dimensions*. The bigger expressive power of Noema doesn't mean that one has to program hypertext, as is the case with ISE/IML, so Noema is proposed as a better solution to the problems encountered in intensional HTML.

1 Introduction

In 1997, the Intensional Programming community was first introduced to the concept of treating the Web as a series of possible worlds, and to web pages as instances of a demand driven data flow stream. Since that time, there have been three core versions of intensional markup for bringing user-determined versioning to web site use: IHTML-1[10], IHTML-2[1], and currently ISE/IML [7]. While the first two versions of the markup language to support versioning were similar, the third was a significant departure. The first two attempted to use a markup approach that would only add a few HTML-like tags to HTML, making versioning accessible to the majority of non-programmer Web page authors.

As the limits of this approach became apparent through various test sites, a more robust programming-like solution was sought. The result was ISE/IML. This approach, while robust, lost for non-programmers the immediate accessibility to build their own page effects, forcing them to rely on prefabricated intensional macros instead.

As the authors of these systems have stated [1],[3], neither the IHTML nor ISE/IML approach is optimal for bringing flexible version control to non-programmer web authors. In this paper, therefore, we present Noema as metalanguage, middle ground approach to the problems posed by both IHTML and ISE/IML. In the following sections, we briefly overview IHTML and ISE to situate the problem space. From here, we introduce Noema and demonstrate how we can use this as a markup interface for versioning primitives which will let authors have more direct, but interpretable control of the versioning functionality.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 228–239, 2002.

2 Overview of IHTML and ISE/IML

Intensional HTML (IHTML) was a first step implementation towards proposing the Web as an intensional set of possible worlds. That is, each site could be treated in the manner of Plaice and Wadge's Intensional Programming [5]. That is, sites, like software, could be represented as versionable components, rendered on the fly to satisfy a user's request for a particular version. Three of the key benefits of creating demand-driven pages are (a) the reduction of duplication/cloning if maintaining multiple versions/localizations of a particular site; (b) the possibility for a wide range of version combinations potentially not anticipated by a site designer, and (c) the concept of "best match" for any given version request.

The first implementation of an intensionalized web, IHTML, took the form of an extended HTML markup. IHTML markup was relatively straightforward for an author to create and for another human to interpret. For instance

```
<a href="page.html"
   version=lang:french%canadian>
french version </a>
```

meant render this particular page with all attributes on the language dimension set to French Canadian- where those components are available. Where French Canadian is not available, default to French.

Simple statements like the above made IHTML immediately appealing. There was however a high cost for creating the version components and labeling them. The markup itself, for things no more complex than simple case statements for offering possible versions of a page became cumbersome.

For instance, the intensional concept would make it possible to create a "slide show" as a series of possible versions of a page. The markup costs are high, however, as the following sample demonstrates:

```
<ISELECT>
  <ICASE version="slide:1">
    <IMG src="sld1.jpg"><P>
    <A vmod="slide:2">Next</A>
  </ICASE>
  <ICASE version="slide:2">
    <IMG src="sld2.jpg"><P>
    <A vmod="slide:3">Next</A>
  </ICASE>
     ...
  <ICASE> I'M DONE AT LAST!!
  </ICASE>
</ISELECT>
```

A programmer would be frustrated if s/he was forced to write in a language that can not take advantage of the fact that all 250 cases above are instances of the same generic template, namely

```
<ICASE version="slide:X">
  <IMG src="sldX.jpg"><P>
  <A vmod="slide:(X+1)">Next</A>
</ICASE>
```

In other words, it was difficult for IHTML to handle simple functionality. To address this problem, a new approach was taken with the development of ISE and IML. ISE is an intensionalized version of the Perl language. As such it is custom-designed to handle any of the functional requirements missing from IHTML. IML was developed as a kind of front end for ISE. It would provide a set of predefined macros (in TROFF) for page authors to use to embed intentional functionality into a page. For instance, Nelson's drop text could easily be embodied as an ISE function through a macro call:

```
.bdrop - 1 Info about fish

The fish will be spawning here in three weeks
[whatever HTML markup the author wishes]

.edrop
```

The result of this call would be

```
>  Info about Fish
```

where the > is a link. Click the > and the page reveals the material under the header, re-orienting the page to the header (-1) of this point (info about fish).

When the author finishes the markup, they run a process to convert the file into ISE. They create a link to the .ise file, such as `"fish.ise"`, and a server-side process renders the ise into the appropriate HTML for a given version request of that file. Any link requests from the page are sent to the server as ise parameter requests causing a new html page to be sent to the client that represents the new version request. Like the original IHTML, ise pages are always represented to a browser as HTML, so they are accessible from any browser. The browser is only limited by the javascript/html or whatever effects the author adds to the page. IHTML and ISE are otherwise platform agnostic.

With macro calls like this available to an author, clearly the weight of having to write Perl (or javascript or for that matter reams of IHTML) is highly diminished, allowing the author to concentrate on their design rather than on programming effects. On the down side, the IML approach does lose some of IHTML's more intuitive clarity and also requires someone proficient in both TROFF and ISE's specialties to create new macros.

Indeed, several papers on IML have lamented the "ugliness" of IML [9],[4] and have expressed a desire to convert it into a more XML-like form for better integration into the markup of the page - recovering some of what was lost from IHTML.

The challenge, therefore, is to find a middle ground between the full-fledged programming functionality of ISE and the benefits of more markup-like integration of IHTML. The solution should allow the author to create customized constructs (missing from IHTML) without escaping from the authoring language (as IML requires by backing up through TROFF to ISE). Such a solution is achievable by consideration of what is to be gained by adding functionality at the markup level. In the following paper, we present one possible solution: Noema. Noema is a hypertext metalanguage that generalizes IHTML in describing intensional hypertext. The three main ideas underlying Noema are: the unification of hypertext and versions, the use of versionable XML-like entities (and the version changes in pieces of a hypertext file) and the implicit dimensions as a substitute for state. We describe each of these concepts below and conclude with a discussion of future work.

3 Introduction of Noema

As demonstrated above, a concern with IHTML is the absence of markup reuse and of textual manipulation of versions. The missing features are not as general as programming functionality, nor do we need the imperative paradigm with states and the like to implement them. However they are very broad to be implemented by defining new constructs, because our ultimate goal is generation of specific markup from generic markup (that is, certain text manipulation facilities)[1].

In this section, we propose a different approach to the problem. The goal for our design is to create a meta-language (not necessary an HTML, or IHTML superset), that is going to encompass the enhanced handling of versions we require:

- Manipulation of versions as hypertext
- Text reuse in a parameterizable fashion
- Some notion of state
- As friendly to the author as possible; markup rather than imperative

In the following sections, we present the design of Noema, such a meta-language. It is important to note here that this isn't a complete presentation of the language. Many design details have been deliberately left out. The focus here is on the ideas that underlie the design and Noema syntax is meant to help demonstrate these ideas in a more concrete way. The ideas under discussion are:

- Use of hypertext to denote versions: We will exploit the tree-like structure of hypertext to express the tree-like structure of versions. In this way, we may use both interchangeably and this results in a nice uniformity when it comes to parameter passing.

[1] Arithmetic may be regarded as text manipulation too

- Versioned entities: Entities are an XML concept missing from HTML and IHTML. With entities one names a piece of hypertext to be used several times in the document. What we are going to define is versioned entities. This makes Noema be to IHTML what XML is to HTML. Versioned entities can be seen also as *parameterizable* entities, because we can invoke any version of an entity we want. Versioning is enough to pass parameters, so we will not use any novel machinery for that.
- Implicit versions as a substitute for state: An invocation of an entity generally implies that we are going to use a specific dimension in our version space as "state". We make the existence of this dimension implicit and this choice makes the language strictly more expressive.

Noema is a metalanguage that describes hypertext. A Noema file prescribes various version-dependent hypertexts. We therefore need two levels in our language: that of meta-hypertext, and that of literal hypertext. Our conventions will be the following:

- A Noema file consists of a declaration section and a meta-hypertext section (divided by `%%`).
- When writing meta-hypertext we may include literal hypertext within braces ({}).
- When writing literal hypertext we may include a meta-hypertext expression preceded by `&` and succeeded by `;` (like XML entities)
- Special characters such as `&` and {} within literal hypertext, are preceded by `&` if they are to be taken literally

3.1 Versions Themselves Are Hypertext

Version space in IML/ISE is currently very advanced, if compared to the first tries in [5]. The current version space (see [7]) supports nesting, as the following abstract syntax shows (V is the non-terminal for version, s is just any string):

$$V \to \epsilon \mid s : V \mid V + V$$

(also assume that s will be equal to the version $s : \epsilon$).

This syntax is readily expressible with markup, under the following translation scheme ($[V]$ is the markup that corresponds to version V):

$$[\epsilon] \;=\; \texttt{<!--EMPTY MARKUP-->}$$

$$[s : V] \;=\; \texttt{<}s\texttt{>}[V]\texttt{</}s\texttt{>}$$

$$V_1 + V_2 \;=\; [V_1][V_2]$$

Versions are a limited form of hypertext because not all hypertext can count as versions under this translation scheme. For example`<A><B/></A>` represents version `A:B`. But `<A><B/></A><A><C/></A>` doesn't count as a version, because the `A` dimension appears twice.

This scheme offers a uniformity, with certain advantages. For example, later we introduce parameter passing to an entity via versioning. Because versions and hypertext are considered the same, we can thus pass pieces of hypertext as parameters. We can also version versions and so on.

Since we express versions as hypertext, our meta-language should have some operators that handle versioning. To that end we also borrow the semantics of most of the original IHTML constructs:

- `cv` is a nullary operator, that returns the current version
- `vmod` is a binary operator. `a vmod b` means version `a` modified by modifier `b` (modifiers are also expressed as hypertext). We may also use it as a unary operator, where `vmod b` means `cv vmod b`. Version modification is exactly the same as in IHTML
- `++` stands for version algebra + and `:` stands for version algebra :
- `(/)` is the vanilla version (or the empty hypertext)
- `'` followed by an identifier denotes that version name, e.g. `'A:'B` stands for hypertext `<A><B/></A>`[2]

Note that the + operator of version algebra is not exactly the same as hypertext concatenation, since for example it is idempotent[3]. Operator `,` denotes pure hypertext concatenation.

3.2 Limited Scope Version Modifiers

We introduce the `@` binary operator whose purpose is to modify the current version within its scope. If `a` and `b` are meta-hypertext expressions, then `a@b` denotes `a` at version `b`. For example,

```
{someHypertext}@(vmod someModifier)
```

changes the current version by `someModifier` so that `someHypertext` is evaluated in the changed version. Similarly,

```
{someHypertext}@(someVersion)
```

sets the version of `someHypertext` into `someVersion` altogether. The difference of `@` and `vmod` and `version` attributes of IHTML is that in Noema we may change the version of an arbitrary piece of hypertext, instead of just one link.

3.3 Versioned Entities

The main problem is to make text re-usable. Here's where the entity idea comes: we name a piece of text, so that we can reproduce the text by using only its name. This exactly what XML entities are doing [2]. But we want more than that: as we've already seen in our examples, the text to be re-used is not exactly

[2] Note that we use the XML abbreviation `<X/>` for `<X></X>`

[3] In general, any version is hypertext, but not any hypertext is version

the same, but rather it depends on parameters. Adding simple XML entities to IHTML won't solve the problem. Instead, we need to make parameterized entities possible. Instead of new machinery for parameters, we prefer to use something already existing: versions. Not only does this avoid the formal clutter by not introducing new constructs that are not needed, it also has the well known advantages of versionning, such as the use of version refinement etc.

Noema introduces entities and entity references (in the XML sense) that have versions. For that matter, we use the `entity` meta-construct in Noema, for the creation of entities in the declaration part of a file:

`entity` *name_of_entity*{*noema_code*}

where in *noema_code* we have a Noema file again (declarations / meta-hypertext etc.).

What is contained in this construct defines exactly what markup is going to be generated when the entity is referenced. The markup depends on a version, so by referencing the right version of the entity[4], we get the exact markup we want. Entity declarations may be nested, but then the inner one is regarded as "local" to the entity definition and is not available out of it. Recursion is allowed. To reference the entity within meta-hypertext we just use its name.

3.4 Other Primitives

Except from what we've shown so far, Noema also supports as meta-hypertext numeric values (that stand for their corresponding decimal representation) and all usual mathematical operations, and it also supports the logical operations `and`, `or` and `not`. These operate on "boolean" hypertext, that is, meta-hypertext that either equals to `'TRUE` or to `(/)`. There are comparison operators, that return such boolean hypertext:

- Hypertext comparisons: `=` and `!=`
- Numerical comparisons: `>=` etc. (numerical equality and inequality coincide with their hypertext counterparts)
- Version comparisons: version equality `~` is weaker than hypertext equality. The appropriate notions of version inequality `!~` and version refinement `[~` and other derived operators such as `[`, `]`[5] and `~]` are also supported

The *conditional hypertext* takes a boolean meta-hypertext expression and two other meta-hypertext expressions. Its semantics is obvious. For example:

```
if cv [~ 'language:'french
then {bonjour}
else {good morning}
```

To get the hypertext within a tag we use `get` as a binary operator. `a get b` gets the value of version `b` within hypertext `a`, for example

[4] We can do that using the `@` operator

[5] Strict refinements

```
('language:'french) get 'language
```

evaluates to `'french`. Or, similarly,

```
{
  <GREETING>Hi!</GREETING>
} get 'GREETING
```

evaluates to `Hi!`. `get` will work on `cv` when used as a unary operator. `strip` is a unary operator that strips off the outermost tag in its parameter.

Operator `linkto` creates a link to a Noema file described by its parameter. We usually write a filename in braces for literacy (but the filename could also be an arbitrary meta-hypertext expression!) and we define a specific version of the file using the `@` operator. `link` is a parameterless version of the operator, that refers to the same file it appears in.

A further abbreviation: Suppose we want to create an entity `a` equal to the hypertext that lies under dimension `d`[6]. We normally include in the declaration part:

```
entity a { %% get d }
```

We may abbreviate this declaration to:

```
par a d;
```

3.5 The Drop Markup Example

The drop text effect could be implemented in Noema (figure 1)

```
entity drop {
  par mode 'MODE;
  par content 'CONTENT;
  par abstract 'ABSTRACT;
  par meta 'META;
%%
  if mode = 'EXPANDED
  then (link @ meta:'HIDDEN):{[-]} , content
  else (link @ meta:'ABSTRACT):{[+]} , abstract
}
```

Fig. 1. The drop-markup in Noema

The few nested `par`'s just give convenient names to the parameters of the entity. The value of dimension `'MODE` can be either `'EXPANDED` or `'HIDDEN`. `'CONTENT` and `'ABSTRACT` contain the two versions and `META` describes the version of the document to be modified, if the user clicks on the link `[+]` or `[-]`. So a "client" of the `drop` entity would do something like:

[6] Thus i.e. we name "formal" parameters within an `entity` declaration

```
drop @ vmod
(  'CONTENT:{Blah blah}
++ 'ABSTRACT:{Blah}
++ 'META:'MODE
)
```

This code says that the state of the drop markup depends on the version dimension `'MODE`.

3.6 Implicit Dimensions as State

The code we created is still unsatisfactory, since we need to write all this information about the `'META` dimension (on which depends whether the markup is "hidden" or "expanded"). It would be nicer if the inclusion of a drop markup control within the document implicitly created such a dimension, already known to the control, which is then able to adjust itself. That among others would spare us the clutter of creating the `'META` dimension ourselves in the previous example.

To achieve this we introduce the *implicit dimension* concept. Each reference to an entity in the document implicitly creates a new version dimension which can be used to change the status of the document. We access this dimension *within the entity definition* by `i(x)` where `x` will be the entity name. A command `this(x)` re-invokes entity `x` *using the same implicit dimension.* The drop markup example is now as seen in figure 2.

The point of this example is, apart from showing the actual Noema code, to demonstrate that it is very easy to write a Noema entity definition. When defining an entity, unlike in IML, the user has at his or her disposal the full power of the language[7]. The language is only one and it is the same for both definitions and invokations of entities. The definition syntax is also considerably less cluttered than the IML counterpart. It is therefore believed that the Noema framework is better in many ways than the IML front-end. It remains to be proved that it is actually so.

Now the client is:

```
drop @
(  'CONTENT:{Blah blah}
++ 'ABSTRACT:{Blah}
++ 'INITIAL:'HIDDEN
)
```

Note that the number of implicit dimensions added is not fixed. This is because the number of actual entities invoked is not fixed in a Noema document: it is a variable that depends on the version, as shown in the code below:

```
entity multiply {
  par num 'NUM;
```

[7] In IML, we either don't have ISE at our disposal, or we have to resort to programming

```
entity drop {
  par initialmode 'INITIAL;
  par content 'CONTENT;
  par abstract 'ABSTRACT;
%%
  if get i(drop) = 'EXPANDED
  then (link @ vmod i(drop):'HIDDEN):{[-]} , content
  else if get i(drop) = 'HIDDEN
       then (link @ vmod i(drop):'EXPANDED):{[+]} , abstract
  else this @ vmod i(drop):initialmode
}
```

Fig. 2. The drop-markup with implicit dimensions

```
  par markup 'MARKUP;
%%
  if num = 0
  then (/)
  else markup ,
       multiply @ vmod 'NUM:(num - 1)
}
```

This construct is given a markup m and a number n and its output is n consecutive copies of m. n may depend on version, so there is no way for the author to determine how many entities s/he has in the document if it contains a `multiply`.

3.7 The Slide Example

Figure 3 is how we do the slide example in Noema utilizing everything introduced so far[8]. We notice that it takes a very short and easy definition to create something impossible in IHTML and very hard in ISE. The difference from ISE lies at the functional-like approach of Noema which makes the definition of the construct more natural and compact. This is also true for the previous examples.

4 Conclusion

Noema as presented focuses on and achieves improvement which the IML community has sought for versioning hypertext:

– First, Noema is an authoring language as opposed to a programming language. It is not all-capable, so it cannot perform tasks unrelated to web authoring and potentially harmful. It is not as involved and complicated as a full-fledged programming language. However, its functionality has not

[8] This example specifies a series of slides found in jpeg files, where the name of the file for slide X is "$prefixX$.jpg", where $prefix$ is a parameter to the `slide` entity

```
entity slide {
  par prefix 'PREFIX;
%%
  {
    <IMG src = "&prefix , get i(slide);.jpg" /><P/>
  } ,
  (link @ vmod i(slide):(get i(slide) + 1)):{ Next }
}
```

Fig. 3. The slideshow in Noema

been compromised. Text reuse, parameterization, recursion and states are supported.
- Versions and hypertext are considered the same in Noema. The flexibility gained by this syntax unification is very important. A version may contain hypertext in it (we saw that in the drop markup example where the `<CONTENT>` and the `<ABSTRACT>` versions are full-fledged hypertext) and parts of hypertext may be used (or compactly written) as versions. Versions can themselves be versioned.
- Noema allows for parts of the versioned document to have different versions. It also allows XML-like entities. The combination of these features offers everything needed for fully parameterized hypertext
- Noema can be viewed as a language that prescribes how a hypertext document is going to be, independent of the target language, that could be for example XML.
- Another idea introduced in this paper, somewhat orthogonal to the previous ones, is the concept of implicit dimensions. We saw that the effect of this is that entities within a document are given state. Since the number of entities invoked in a Noema document depends on the version, the version space is not definable in authoring time.
- Noema can be a substitute for TROFF macros. Its syntax is more readable than TROFF. In the Noema framework, there is but one language. In IML, if one needs more than what is already implemented, they might need to resort to a second language, ISE.

The next step should be the creation of a formal specification for the language and an efficient implementation. From the language design point of view there is more work to be done to study the strengths and weaknesses of the language and to discover new ways of using it or new properties of interest and areas of improvement. As far as applications of Noema are concerned, it seems that it has potential to be used as a versioning tool for XML, for example XML components of software systems, taking versioning back to its roots, where it was designed for software component versioning.

References

1. G. D. Brown. Intensional HTML 2: A practical approach, MSc Thesis, Computer Science Department, University of Victoria, 1998.
2. World Wide Web Consortium. Extensible markup language (XML). Available on-line at http://www.w3.org/TR/REC-xml.
3. m. c. schraefel and W. W. Wadge. Two cheers for the web. In M. Gergatsoulis and P. Rondogiannis, editors, *Intensional Programming II*, pages 31–39. World Scientific Publishing Co. Pte. Ltd, 2000.
4. m. c. schraefel and W. W. Wadge. Complementary approach for adaptive and adaptable hypermedia. In *Intensional Hypertext. 3rd International Workshop on Adaptive Hypermedia*. Springer-Verlag, 2002. Forthcoming.
5. J. Plaice and W. W. Wadge. A new approach to version control. *IEEE Transactions on Software Engineering*, 19(3):268–276, March 1993.
6. P. Rondogiannis and W. W. Wadge. Intensional programming languages. In *Proceedings of the First Panhellenic Conference on New Information Technologies (NIT'98), Athens, Greece*, pages 85–94. National Documentation Center of Greece, October 1998.
7. P. Swoboda. Practical languages for intensional programming. MSc Thesis, Computer Science Department, University of Victoria, 1999.
8. B. Wadge, G. Brown, m. c. schraefel, and T. Yildirim. Intensional HTML. In E. V. Munson, C. Nicholas, and D. Wood, editors, *Principles of Digital Document Processing*, volume 1481 of *Lecture Notes in Computer Science*, pages 128–139. Springer-Verlag, 1998.
9. W. W. Wadge. Intensional markup language. In P. Kropf, G. Babin, J. Plaice, and H. Unger, editors, *Distributed Communities on the Web*, volume 1830 of *Lecture Notes in Computer Science*, pages 82–89. Springer-Verlag, 2000.
10. T. Yildirim. Intensional HTML, MSc Thesis, Computer Science Department, University of Victoria, 1999.

Sharing Social Recommendations: Towards a Social Portal

Tim Mansfield, Nigel Ward, Markus Rittenbruch, Gregor McEwan,
José Siqueira, and Anthony Wilkinson

Information Ecology Project
CRC for Enterprise Distributed Systems Technology
University of Technology Sydney, University of Queensland and Monash University
timbomb@dstc.edu.au
http://www.dstc.edu.au

Abstract. The Information Ecology project at DSTC is constructing a Social Portal. This article explains what we mean by a Social Portal, what user needs we believe we are serving by building one, what research goals we think we are serving and how we intend to go about it.

Keywords. Information retrieval, social navigation, recommender systems

1 Introduction

The Information Ecology project at DSTC is an interdisciplinary programme investigating how to improve the interconnectedness of the online experience. One of our goals is to enable software to better exploit the broad context (both within the computing environment and beyond it) of the execution of a user command. We intend to investigate ways in which software can appear more integrated with its environment by exploiting context more effectively.

Several kinds of context are initially appealing and this paper describes initial work in studying one kind: the patterns in people's social interaction with each other. The complexity of studying human interaction is great enough to have spawned several entire academic disciplines[1], so we have attempted to narrow our scope. We are interested specifically in what can be done with information about the people with whom our user *communicates electronically*.

Our initial approach is to gather that information by providing an application which supports communication with a list of contacts and use that as a way to capture information. Based on this data we want to answer several research questions that are outlined in Section 8. For this to be useful as a research tool, we need an application which people will use for a significant proportion of their communication, therefore:

- it should serve a known need
- it should be more effective than existing solutions
- it should be primarily web-based to minimise the barriers to adoption.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 240–252, 2002.

These criteria led us to the notion of a "social portal". Section 2 defines this notion. Section 3 describes how this relates to user needs. Section 4 offers a critique of existing kinds of portals. Section 5 suggests how socially contributed information may be included in a portal design resulting in a kind of *social portal*. Section 6 contrasts the social portal with web-based shared workspace systems. Section 7 describes the design of the system and the field tests. Section 8 outlines the research we want to conduct when we have an appropriate user community.

2 What Is a Social Portal?

Portal sites such as MyYahoo![1] or Lycos[2] or MyNetscape[3] attempt to pull *all information of interest* to the user together in one place. This information is usually organised as *channels* of information on some topic. Users can typically personalise the channels they see in a portal. For example, users can choose to see channels from newswire services such as Reuters[4] alongside stock portfolio channels, TV listing channels, horoscopes, weather, and so on.

By putting *all information of interest* in one place, these sites serve the user by providing a convenient starting place for accessing information of interest and keeping up to date with that information.

Although portal sites aim to put everything the user is interested in together in one place, in fact many interesting sources of information are omitted. Portals may include newswire news, but they rarely put these next to a channel about your sister's family news or a channel containing discussion of your company's new strategic direction.

We are building the Social Portal to investigate portal-style presentation of information from social networks. Rather than solely relying on general channels that may meet the user's information needs, we are building a system (described in Section 7) that also uses social context to recommend information.

Individuals can use the system to send messages to social contacts such as colleagues, friends and family. The receivers of this information can personalise the portal to see the contributions of their friends alongside traditional portal channels.

We wish to be clear that we are discussing an evolving, web-based prototype. The initial version simply focussed on sending channels of messages to social contacts. It also included a simple interface for personalising your view of received channels. Later versions will include access to traditional portal information sources.

[1] `http://my.yahoo.com`
[2] `http://www.lycos.com`
[3] `http://my.netscape.com`
[4] `http://www.reuters.com`

3 How Do Portals Serve the User?

Portals provide customisable interfaces for displaying multiple *channels* of *information items* on some topic. Users are given control over which channels they see and can customise how they see those channels.

Portals often provide a large number of fixed channels for the user to choose from. The sources of items in these channels range from newswire services such as Reuters, other portals, automated information feeds such as stock price reports, individual authors (newspaper column style), or professional editors collecting information of interest.

Portals attempt to pull *all information of interest* to the user together in one place. The popularity of portal sites suggest that having even some of the user's information streams in the one place serves the user well. Portals give the user a convenient starting place for accessing information of interest and provide a convenient way to keep up to date with that information.

A less obvious advantage of the *in one place* approach is that such systems can understand the user better. Recommender systems[2] are portal-like tools which recommend information to the user based on their understanding of the user's information interests. This understanding is built up by observing the information the user manipulates. If users access all of their information through the one place, these systems can make better observations and hence more targeted recommendations.

We also believe that the portal presentation style can help users better structure their information space. Some portals allow channels to be constructed from multiple information streams. For example, MyYahoo! allows the user to construct a European news channel by selecting information sources on German, Swiss, Danish, Spanish, French, etc news sources. A social portal would allow construction of channels with information from existing portal channels *and* other people you know. For example, the user could construct a "Corporate Strategy" channel consisting of information from the corporate news feed, their organisational unit's discussion forum and personal emails on the topic.

4 A Critique of Portals

Portal channels serve the whole channel audience. Decisions about what items are recommended to the channels are made based on a profile of the channel audience. The channels serve the group rather than the individual. This results in the recommendations within a channel being high-level, broad information that does not always *target* any individual within the channel audience well.

Horizontal portals, also called mass-market or broadcast portals, such as Yahoo! or MyNetscape particularly suffer from this targeting problem. They provide channels of broad interest to large demographics (such as newswire services).

Vertical portals or community portals address the targeting problem by providing access to fewer channels on a particular topic or targeted to a particular

audience. For example, Slashdot[5] provides "News for Nerds - Stuff that matters" and Redherring.com[6] is an online companion to Red Herring magazine, providing information on technology business.

Although vertical portals have more targeted channels, they still rely on a broad understanding of an audience and so do not always contain exactly the information and only the information of interest to a user.

Enterprise portals are portals that operate within an organisation. They address the targeting problem with channels containing information drawn from within the organisation. They can provide access to more targeted information such as corporate calendars, internal memos, and even personal information sources such as email and personal schedules.

Enterprise portals only contain organisationally accredited information. They tend to reflect a static view of the organisation. There is rarely support for information exchange between individuals or groups that have no official existence.

Fundamentally, the shortcomings of existing portals arise from the portal having limited knowledge of the individuals that they serve. The only knowledge of the users is from generalisations of the demographics, whether it be an editor or author's understanding, or a machine's statistical understanding of the audience.

In summary, portal software works well as a presentation paradigm for showing all the information that the user needs in one place. The portals that we have investigated have limited representation of information sources. An important source of information that we feel is neglected by current portals is the people you know.

5 Adding Social Information

Users already receive information of interest via their social relationships. Informal observation suggests that this is via email, either personally or on group mailing lists, via instant messaging systems such as ICQ[7] or Yahoo! Messenger[8], via discussion fora such as Usenet News or Topica[9], and occasionally via references to web pages containing news, writing, pictures and links to sites the author thinks are interesting.

Sharing information, particularly digital information like text, digital pictures and web links, is a process particularly amenable to digital media streams such as these. One main issue for the receiver is that it is difficult to see all of this information *in one place*. Although many portals support access to email, discussion fora and messaging systems, they do not provide a unified view of these media streams. Traditional portals do not allow users to place new email messages in a portal channel alongside a Reuters world news channel, alongside

[5] `http://www.slashdot.org`
[6] `http://www.redherring.com`
[7] `http://www.icq.com`
[8] `http://messenger.yahoo.com/`
[9] `http://www.topica.com`

a channel of ICQ messages from their distributed colleagues, and so on. Instead the user must move into different sections of the portal to see these different media streams.

The prototype we are building will eventually let the user see much more information of interest *in one place*, including information from social contacts.

6 Portals and Shared Workspaces

A social portal has some similarity to a web-based collaboration system such as BSCW [3]. Web-based collaboration systems provide users with the ability to create different workspaces, to invite other users into their workspaces and eventually to share information of interest such as URLs amongst each other. They therefore potentially mimic social portals whilst being functionally more comprehensive. In spite of these similarities social portals differ from such web-based collaboration systems in two major ways.

First, shared workspaces and social portals are based on different metaphors. Shared workspaces in general, implement a notion of space. In order to manage and structure access to shared information and interactions these systems provide different views on shared resources. Systems based on the commonly used room metaphor, such as Rooms [4], Moksha [5], TeamRooms [6] or DIVA [7] allow shared artefacts and activities to be located and performed in distinct locations within the system. Other systems based on closely related structural metaphors e.g. folders in GroupDesk [8] or containers in PoliAwac [9] provide similar functionality.

While this approach is suitable for supporting the different specific needs of working groups and single users it is rather too complex when it comes to handling the single task of sharing recommendations in a lightweight and flexible manner. Social portals however follow a portal metaphor, trying to bind as much relevant information as possible in one place. While the range of functionality supporting the cooperation of users within different groups is somewhat limited in comparison to shared workspace systems, social portals allow users to grasp relevant information without the need to browse between different workspaces. A one-place approach seems to be particularly suited to representing ephemeral information like recommendations or other frequently changing and dynamic information.

Second, the structured nature of shared workspaces often restricts the customization of their user-interfaces. While these systems allow for a wide variety of adaptations of workspaces on a functional level (access rights, group members, media types, etc.), the appearance at the user-interface is commonly prescribed. Portals in comparison genuinely support adaptations of the appearance of the system. Since all information is gathered basically in one place the selection which information has to be presented as well as where it has to be presented becomes crucial.

To sum up, the subdivision of work into shared workspaces imposes additional navigational effort on the user and often adds restrictions to the customization

of the user-interface. This additional effort may be well justified in view of the complex task of supporting cooperation in multi-group environments. In order to support the sending and receiving of recommendations, social portals seem to be a leaner approach which allows users to perform this particular task more flexibly and efficiently.

7 Design Approach

Our design approach is both *synthetic* and *user-centred*.

Synthetic, because we designed the initial version of the Social Portal by immersing ourselves in as many existing web applications as we could. The design that resulted is a synthesis of functions that we had seen in various systems that fill a gap that we perceived.

Our approach needs to be user-centred because we are grounding our research activities in data obtained from real social networks. The system has to capture a large amount of social network information to provide the necessary data. Therefore, it needs to be an application that a lot of people will use regularly, in place of their existing methods of sharing information through their social networks.

We sought an initial, plausible design based on our own intuitions and worked toward producing a viable implementation which we could field test.

Following the results of the field test we sought to improve the design by engaging our user community more closely in the design process by using the *Extreme Programming* methodology[10].

This section outlines that initial design, the conduct of the first field test and our decision to use Extreme Programming.

7.1 Initial Implementation Design

The initial system was based on the common portal metaphor of an online newspaper. Rather than receiving news items from a wire service like Reuters in this newspaper, users would receive recommendations of web pages from other users. Like many online newspapers, the Social Portal organises items into channels, one channel for each topic contributed by a given user.

The reverse-side of this design is that each user can make up channels about topics they would typically share and send recommendations to friends or colleagues using these channels.

So, the initial version was based on these two basic notions: recommendations and channels.

Recommendations had a title, a URL (for the thing being recommended), a description and a sender and date. Users soon worked out that they could omit the URL and just use recommendations to send a text message as a news item.

Channels are a conduit from their sender to a group of receivers. Each channel associates a set of recommendations on a common topic (in the opinion of the sender) with a set of receivers.

Each receiver had a page consisting of all the channels they had been sent. They had the means to rearrange the order of the channels in a two-column layout. Figure 1 shows the page of channels. There are three channels shown: "Interesting Systems" and "Research Stuff" recommended by the user timbomb, and "Database" recommended by the user eggplant. The visual design of this page was heavily influenced by MyYahoo!.

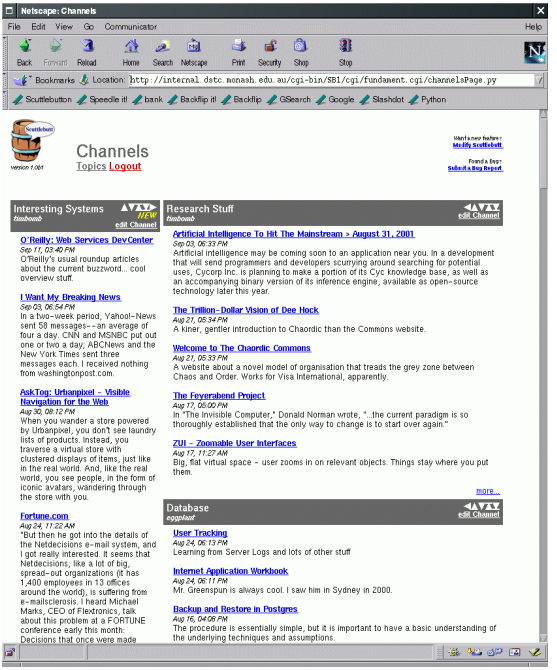


Fig. 1. The Channels Page

In addition to receiving recommendations via the web page, users could choose to receive asynchronous notifications via email or via an asynchronous notification system, the Elvin tickertape [11], produced at DSTC.

Before describing the field test, it seems prudent to summarise some of the things the system did not do. This list is created with the benefits of hindsight, but their omission may not be obvious.

- The users could not send any recommendations without creating a channel.
- The system had no facilities for communication other than the channels, so users could not discuss recommendations they had been sent and giving feedback to the sender was difficult.
- Users could not add themselves or each other to channels they did not own.
- There was no way to discover what channels were being published.

Although it was obvious to us that these omissions would cause problems, we felt it was important to settle on a simple design and be led by our users in judging which omissions were most critical.

7.2 Initial Field Test Conduct

The initial field test was performed within our organisation. In order to involve users in the design process as early as possible we used an early prototype which was reasonably stable, but had limited functionality.

Subjects. We designed the system with a "viral" model of promotion. Whenever a recommendation was sent to someone who was not a registered user, that person would be sent an email inviting them to register. We extended this virus metaphor, so that new users had to be invited before they could register.

We were interested in how far the system would spread within a community if we released to a small sub-group within that community.

We initially targeted a group of 10 users within our own organisation, across several different groups within the organisation. The group consisted of several research staff, one administration worker, two marketing staff and a number of programmers.

Deployment. We installed the system within the organisation's internal network and placed restrictions within the system to prevent users from sending recommendations to anyone outside the organisation.

In addition to the basic system functions we added help functions and feedback forms for bugs and feature requests. All users were encouraged to report bugs and communicate requests for new functionality via the system. We choose electronic feedback in particular since our users were distributed over three different sites within Australia.

The help functions and a communication channel from the developers to the users (announcements) were represented as channels within the system and provided to every user.

We conducted no initial user training in order to test whether the system usage was intuitive.

Data Gathering. We gathered data in three forms:

1. **System Logs** – The system logs every action the user takes in the system (users gave consent to this before registering),
2. **System Feedback** – the feature request Wiki and the bug tracking system collect some concerns that users felt strongly about,
3. **Semi-structured Interviews** – we conducted face-to-face and phone interviews with six users after they had been using the system for two weeks

Results. In this initial field test, we were effectively conducting an acceptance test: was the system adequate to support the user group's information sharing needs? Could it form the basis of a *popular* system we could use to collect the data that we need?

Consequently, we mostly used the system logs to examine:

- how often users used the system
- how many registered to use the system.

The results of that examination appeared to be that initially the system enjoyed a brief burst of popularity, there were a lot of registrations and a large proportion of the users logging in daily. Interest soon waned and use of the system became very irregular, with only one or two users logging in each day.

The graph in Figure 2 shows how many users logged on each day during the trial period in comparison to the total number of registered users. The burst of initial activity can be seen in comparison to the erratic use of the system towards the end of the trial period. The small, regular periods of no activity from the users correspond to weekends.

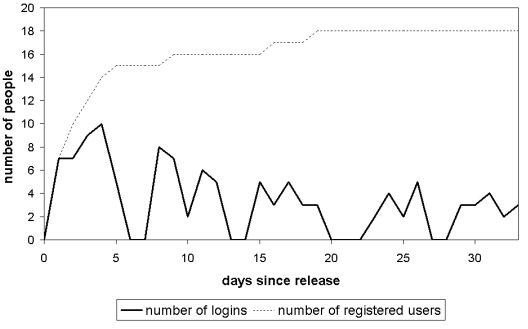


Fig. 2. A graph of system usage.

We took this to mean that the system essentially failed the acceptance test. Examination of the system feedback and the design of the interviews were aimed at discovering why.

Many comments from users concerned usability improvements: more effective ways to layout and manipulate the channels page, better conceptual structure for pages, clearer layout and language for less frequently used functions. Some users requested the ability to send recommendations to *ad hoc* groups without defining a channel first.

But the most striking theme in the user feedback was the desire to have more communication with other users. Users requested features such as:

- the ability to find users and channels that were previously unknown, effectively using the system to extend their social network,

- the ability to give feedback, via comments and via ratings, to the sender of a recommendation,
- the ability to send recommendations to channels started by other users, effectively turning the broadcast-style channel into a discussion forum.

In reflecting on the feedback from this initial field test we decided that the fundamental design of the first implementation contained serious errors, so we set to redesigning it. Rather than attempting a complete redesign and redeployment, we chose to keep the system in service and roll the new design in piecemeal.

Once we believe, based on our feedback, that the users in this initial group are served adequately by the system we intend to extend our field-testing to new user communities and continue gathering feedback.

In responding to the feedback from the field test, we decided to adopt the Extreme Programming methodology (XP) [10], which guarantees the close involvement of users and efficiently focuses on specific user needs.

7.3 Adding Extreme Programming to the Development Process

XP has several implications for the way users are involved in the development process. XP is characterized by a continuous process of planning, designing, coding and testing. Users have a customer's role within that process. They are an inseparable part of the design activity.

Users continuously trial successive versions of the system while it is under development. Based on their experiences with the system, users write "user-stories" to state problems or wishes, and optionally provide suggestions for solutions. The planning for the next release of the system is based on these user stories. Users (in their customer role) and developers meet regularly in order to discuss the further development of the system. Developers outline the coding time available until the next release. Customers then negotiate between themselves as to which features or problems they wish to have integrated or solved within the next system release. After the specified user stories have been implemented, customers take part in a second feedback loop - the system testing. Discussions between the developers and their customers establish whether the requirements have been accommodated by the system. In the event of customer dissatisfaction, the task will be resumed in the next iteration.

XP is a practical approach resembling several participatory design methodologies [12] such as rapid prototyping or evolutionary software design [13]. It attempts to diminish the gap between requirement gathering and modelling by actively integrating the user into an iterative design process. We have enhanced the original methodology in several aspects. We introduced a design-customer role and a new-user customer role to address two common problems that arise during participatory design processes.

The design-customer suggests modifications that are based on research or design knowledge. The role was introduced to address the common problem that users, to some extent, suggest functionality that is "well-known" from other applications. The design-customer role thus enables the integration of new design

ideas into the process to give users new options for solving particular problems. The role can be performed by several actors at the same time and is negotiated among these actors.

The new-user customer role addresses the problem that users who are involved in the design process, gradually become designers themselves. While this can be beneficial in some respects, these users then lose their ability to see the system from the perspective of a new user. The new-user role is performed by an actor who continuously gathers usage data from new users and feeds the resulting user stories into the design process. The ongoing involvement of experienced and new users, as well as the integration of design aspects provides a well-balanced design environment to suit a broad user base.

We hope that these changes to our development process will improve the usability of the system and its suitability for the needs of our target user groups. This in turn helps support our longer-term research goals by providing us with a sustainable user community who are strongly involved in the system's evolution.

8 Future Work

The long term goals of the Information Ecology project are to study how broad context can be used to improve a person's information experience. The research and development reported in this paper is aimed at capturing and using social context to this end.

To gather useful data, the tool must be a popular application that supports and uses social networks. We intend to improve the popularity of the tool in two main ways: by increasing tailorability and by including the ability to access social recommendations made using other tools.

Because portals gather information in the one place, tailorability of information presentation becomes a critical user issue from the outset. Our prototype already provides some presentation tailorability: channels can be moved around on a page, and the user can choose how they are notified about new information. We intend to investigate moving presentation tailorability outside the system by allowing social portal channels to be embedded in any HTML page. This may be achieved by offering users purpose-built HTML that can be embedded in existing web pages, or by providing the social portal as a web service using one of the emerging web service protocols [14].

Current user feedback suggests that many users make social recommendations using existing systems such as email, SMS, chat tools and so on. Some users have indicated that they would find the prototype system more useful if it provided access to these other vectors of social recommendation. We intend to investigate the integration of social recommendations from these systems into the prototype. We wish to be clear however that we do not intend to replace these existing recommendation tools. Rather we wish to improve the popularity of our tool by providing access to these other types of social recommendation.

Once we believe, based on our user feedback, that we have developed a popular application that supports social networks we intend to instrument the tool

to gather data on how, when, and to whom people make recommendations using the portal. Based on this data we intend to investigate a number of research questions, including:

- Portals attempt to provide the user with a unified view of all information of interest in the one place. Is it possible to provide a unified view of socially recommended information alongside traditional portal channels? What other sorts of information can be effectively viewed in a portal channel model?
- Can we use the system's understanding of social networks to improve the information seen in the portal? That is, can social context be used to improve recommender systems[2]? Can recommender systems be used to improve social context or cohesion?
- How do social networks change over time? How do they relate to other collections of people, such as organisational units or online communities[15].
- Can we support emergent groups in a system that does not explicitly encode the notion of group? Can we gradually introduce notions of group into the system, guided by user requirements and our grounded design process?

With these research questions as guides, we intend to introduce two major pieces of functionality into future versions of the system: integration of traditional portal information and increased support for groups.

Providing access to traditional portal information sources is essential for addressing the first of our research questions. We hope it will also improve the popularity of the system. The web portal community already freely syndicates a wide variety of portal information using the RDF Site Summary (RSS) format [16].

Our experience building other CSCW systems [17,3] was that the notion of group embdedded in those systems often did not reflect the reality of the groups using the system. For that reason, our prototype deliberately does not have an embedded notion of group. Instead the prototype allows communication between individuals who may or may not be in groups. We wish to investigate co-development of a system with our user community that gradually supports more and more specific and cohesive notions of group. We hope that our grounded design process can help achieve this goal.

Acknowledgements. 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 Industry, Science & Resources).

References

1. Greenberg, S.: Computer supported cooperative work and groupware. Academic Press, London (1991)
2. Resnick, P., Varian, H.R.: Recommender systems. Communications of the ACM (1997)

3. Bentley, R., Horstmann, T., Sikkel, K., Trevor, J.: Supporting collaborative information sharing with the world-wide web: The bscw shared workspace system. In: Proceedings of the 4th International World Wide Web Conference. (1995)
4. Henderson, D.A., Card, S.: Rooms: The use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. ACM Transactions on Graphics **5** (1986) 211–243
5. Ramloll, R., Mariani, J.: Moksha: Exploring ubiquity in event filtration-control at the multi-user desktop. In: Proceedings of the Work Activities Coordination and Collaboration (WACC'99), ACM Press (1999) 207–216
6. Roseman, M., Greenberg, S.: Teamrooms: Network places for collaboration. In: Proceedings of the Conference on Computer Supported Cooperative Work. (1996) 325–333
7. Sohlenkamp, M., Chwelos, G.: Integrating communication, cooperation and awareness : The diva virtual office environment. In: Proceedings of the Computer Supported Cooperative Work Conference (CSCW '94), ACM Press (1994) 331–342
8. Fuchs, L., Pankoke-Babatz, U., Prinz, W.: Supporting cooperative awareness with local event mechanism: The group desk system. In: Proceedings of the Fourth Europeean Conference on Computer Supported Cooperative Work (ECSCW'95), Dordrecht, Kluwer (1995) 247–262
9. Fuchs, L.: Area: A cross-application notification service for groupware. In: Proceedings of the Sixth European Conference on Computer Supported Cooperative Work (ECSCW'99), Kluwer Academic (1999) 61–80
10. Beck, K.: Extreme Programming Explained: Embrace Change. Addison Wesley (1999)
11. Fitzpatrick, G., Mansfield, T., Kaplan, S., Arnold, D., Phelps, T., Segall, B.: Augmenting the workaday world with elvin. In: Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work, Kluwer Academic Publishers (1999) 431–450
12. Schuler, D., D. Namioka, E., eds.: Participatory Design: Principles and Practices. Lawrence Erlbaum Association (1993)
13. Floyd, C., Züllighoven, H., Budde, R., Keil-Slawik, R.: Software Development and Reality Construction. Springer, Berlin, Heidelberg (1992)
14. Consortium, W.W.W.: Xml protocol activity. `http://www.w3.org/2000/xp/` (accessed Dec 2001)
15. McArthur, R., Bruza, P.: The abc's of online community. In: Proceedings of First Asia Pacific Conference on Web Intelligence. Lecture Notes in AI, Pittsburgh, PA, Springer-Verlag (2001) to appear.
16. O'Reilly, Associates: O'reilly network rss devcenter. `http://www.oreillynet.com/rss/` (accessed Dec 2001)
17. Mansfield, T., Kaplan, S., Fitzpatrick, G., Phelps, T., Fitzpatrick, M., Taylor, R.: Evolving orbit : a progress report on building locales. In: Proceedings of the Group 97, ACM Press (1997) 241–250

Enabling Technologies for Communities at Web Shops

Till Schümmer

FernUniversitaet Hagen
Computer Science VI – Distributed Systems
Universitaetsstr. 1, D-58084 Hagen, Germany, +49-2331-987-4371
Till.Schuemmer@fernuni-hagen.de

Abstract. This paper identifies needs for communities at web shops. Different types of communities are classified and rated according to their value for e-commerce sites. Three virtual stores and one external instant messaging and presence tool are observed regarding their community support. All these solutions have deficits in means for establishing the community (partner finding) and starting direct synchronous interaction. The GAMA-Mall system, which is presented in the second part, aims to fill this gap. It is a prototype that enhances partner finding and collaboration in a virtual bookstore like Amazon.com.

1 Motivation

This paper proposes an architecture that supports online communities between customers of a web shop. Although many e-commerce web sites have included basic community support, such as newsgroups or contact facilitation (using electronic mail), it is not yet reasonably stated why online communities are important in the area of business-to-customer e-commerce.

One possible answer can be drawn from findings of history. Hardach and Schilling have provided a cultural history on market places [12], which shows the role of social interaction in commerce. Besides the exchange of goods, people exchanged ideas and information on these markets. Consultancy and politics took place at market places as well as scientific and philosophic discussions.

Along with the evolution of financial markets, the industrialization, the standardization of goods, and the specialization of dealers, it became less important that the goods were present, to sign a sales contract. In addition, the improved postal system reduced the need for face-to-face meetings. Thus, the social factor started to vanish from the markets. Open-air markets became rare and were replaced by small stores in the villages.

Virtual market places reduce stores mainly to the presentation of goods. Other customers and salesmen are no more co-located in one store. They don't feel the presence of others and social contacts are not established in the store. Even though many customers prefer the anonymity of virtual shops, there are also customers who would prefer to meet other people in the virtual store.[1]

[1] Evidence for this group of customers is given in Section 3.2 of this paper.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 253–265, 2002.

We propose to compensate this lack of social contacts and social interaction by means of virtual online communities. As Preece points out, many "owners of [e-commerce] sites believe that online communities serve the same function as the sweet smell of baking cakes does in a pastry shop." [18]. Actually, this is the selling point for online communities in the context of electronic commerce: they should "increase stickiness (customer loyalty), [and] viral marketing (word of mouth), reduce [the] cost of customer acquisition, and drive higher transaction levels." [26]

The paper consists of two major parts: The first part will provide an analysis of community types and their applicability for electronic commerce. In the second part, a system architecture is proposed, which goes beyond current e-commerce solutions and enhances the support for communities with respect to contact facilitation and synchronous interaction. The resulting system architecture is explored and applied using the virtual bookstore www.Amazon.com as an example application.

2 Different Types of Communities

Preece [18] defines online communities as a set of "people, who interact socially as they strive to satisfy their own needs or perform special roles, such as leading or moderating." These people share interests or needs, which make up the purpose of the community. They use group policies such as rules and assumptions to guide social behavior and computer systems to mediate the interaction.

Taking this definition, an online community is much more than just keeping in contact by means of e-mail. If e-commerce sites should be turned into community sites, then they have to assist customers to find common goals or purposes, make them explicit and feel as a part of a the community. There have to be ways of interacting with other customers and sensing their presence. In other words, customers should not feel, as if they were browsing an empty shop [16]. And finally, there has to be an incentive to return to the community (otherwise, there is no social binding).

An increasing number of e-commerce companies has realized that virtual communities can become very binding, if they reach the phase of social involvement (e.g. [24]). According to Chapman [7], frequent interaction among the community members is very important to reach this stage of social involvement.

There are different classifications of communities. Most classifications distinguish at least *communities of interest*, *communities of purpose*, and *communities of practice* (cf. [6,7,16]).

Members of a *community of interest* share the same interests in a topic (and often a common background). Examples are discussion groups on television shows or people interested in planets of the solar system. Some authors (e.g. [6]) also define *communities of passion*, which are very close to communities of interest. The difference is that the members are more involved in the community's topic up to the point where they become passionate advocates. Ac-

tually, a community of interest can become a community of passion. An example is the discussion group on a TV show that became a fan club of the show's host.

Communities of purpose consist of members who share a common (short term) goal. For instance customers at a virtual bookstore share the goal of finding and buying books. They all have to go through the same process (i.e. selecting the item and checking out) and they can help one another reaching the goal. Thus, the community of purpose has a functional purpose and it may dissolve after the goal is reached. In contrast to communities of interest, the members don't necessarily share the same interests and therefore, they are not likely to start activities that exceed the community's purpose [6].

If the members of a community share a common profession, they are called *communities of practice*. Their members reflect on the way, how they perform their tasks and enhance their ways of work in a community learning process. Since the community's topic is the member's profession, members are normally highly involved in such communities. Concrete communities of practice are for instance Smalltalk programmers who meet in a Smalltalk users group to shape the process of programming. In some cases, it makes sense that the community is established by the merchants. Given the example of the Smalltalk users group, the main vendor has been heavily involved from the beginning. The well known experts are employees of the company and their involvement in the online discussions is beneficial for both parties (customers and vendor): the customers can get help from the experts and the vendor can steer the discussion regarding his business plans (carefully up to a specific degree; too much advertising would put off the customers).

Marathe [16] adds another type of community: a *community of circumstances*, which is defined by common circumstances such as current life situations, e.g. children reaching the age of puberty. Interaction in these communities is often personally focused and third parties are not involved in the community. Therefore, these communities may not easily be influenced, but they can lead to a strong binding to a seller, if this seller is present in important life stages.

Each of the communities is valuable for customers and vendors in an e-commerce setting. Especially the community of passion and the community of practice are of high value for electronic business, since they have a very strong binding and promise strong customer needs. For a very specialized market it is very promising to engage in a community of practice. Merchants with a very broad range of goods (and therefore no common practice) should rather aim to establish communities of passion or communities of interest (with a product centered focus).

3 Interaction and Communities in Existing Shops

After the different types of communities were discussed in the last section, they will now be related to existing e-commerce sites. For that reason, three virtual stores that support communities are considered. A short discussion on the appli-

cability of external solutions and a set of requirements for communities at web shops conclude the first part of this paper.

3.1 Virtual Stores

The virtual bookstore at *Amazon.com* aims on establishing communities of interest. With each displayed item, the system shows what other users who also bought this item have also bought. The displayed item serves as a basis for finding peers who share the same interest in the item. The peer's shopping history is then used to provide customized advertisements [15]. The only information that end users gain from the peer shoppers is the recommendation of articles. There is no information revealing the identities of peer shoppers, what prohibits direct communication between these shoppers.

The only way to get more involved in the community, is to create reviews. These reviews are attached to the item and (if the user has provided a profile) can serve as an access point for tighter communication. The communication has to take place using external tools, since Amazon.com does not provide communication channels between the users. The next step in involvement is the creation of a personal booklist and a personal profile. Users enter a description of their (virtual) self and comment the books they like. They can also create buddy lists at their personal page. The pages may contain e-mail addresses and shoppers can get in contact with this information. However, the problem is that the search for other (possibly) interesting users is decoupled from the search for products.

To reach a better support for a community of interest (or even a community of passion), the pages would need means for direct (synchronous) interaction between participants. These can include communication tools (such as a chat), collaborative review facilities, or a collaborative browsing engine. There is also a lack of presence information. All pages seem empty when entering the store.

The second hand bookstore *justbooks.com* allows users to sell their used books. Sellers enter information about the book often containing a personal comment. Customers can then search for the book and read the seller's recommendation. Both, customer and seller, share a common interest, since the seller has read the book before. If the addresses of the sellers would be revealed on the page, they could establish tighter cooperation regarding the topic (such as consultancy). The problem is that justbooks generates revenues based on the books that are really sold. If the company would encourage direct communication between sellers and potential customers, it is very likely that they will trade the books without paying justbooks any fees. Therefore, there is no intended way to form a community around specific books or booklists.

Identities of customers and sellers are revealed, when the book is sold. At this time, they can exchange messages directly. In some cases, the following interaction may exceed the intended exchange of delivery instructions: for instance, users can start discussions on the book's content or customers may ask the seller for other interesting books in the same area. Up to now, justbooks forms a community of purpose. But the two examples show that it has the potential to form

a community of interest, if the borders regarding personal communication are removed.[2]

The virtual clothing shop *LandsEnd.com* provides tools that allow two or more customers to browse through the shop together. Therefore, one user opens the collaborative browsing tool (which is provided on the web page) and starts a new browsing session. If the second user wants to join the session, he enters the session key that he got from the first user (exchanged by an external communication system). From then on, all browsing activities that take place in the browser, from which the collaborative session was launched are coupled. Whenever one user follows a link, this navigation is also performed in the other user's browser. To communicate, the users can use an integrated chat tool.

LandsEnd includes very expressive tools for synchronous collaboration, but the collaboration requires customers to know one another in advance. Thus it does not encourage community building: When combined with integrated means for meeting other customers or forming new groups, this tool could strengthen a community because of the shared experiences.

3.2 External Solutions

Besides integrated solutions, there are external solutions such as instant messaging tools that help customers to be aware of other users. While traditional instant messaging tools filter awareness information based on buddy lists (e.g. AOL Instant Messenger [1] or ICQ [13]), there are instant messaging solutions that also use the users current position in the web. For instance Odigo [17] is an instant messaging solution that shows, which other (possibly unknown) users are at the same page or at the same web server.

The Odigo client was used to conduct a short survey and find out, whether or not users of the tool would like it to be contacted by visitors of the same bookshop, they were currently visiting. For this purpose, an Odigo client was launched and in the browser was pointed to the welcome page of Amazon. Odigo then noticed other users, who also entered the Amazon server. These users were contacted with a short introduction saying that this was a survey and asking them, whether they could participate. 42 (48%) of 86 contacted users agreed to answer the questions. More than 20% of them were not aware that other users could see their current position in the web. They used Odigo because it combined different instant messaging services and didn't know about the location logging.

One could observe a very wide range of activities: Some users were looking for movies, some for music, and others shopped for books. The different topics did not match for any of them. To the question, whether they would talk with

[2] At the submission time of this paper, justbooks.com has merged with abebooks.com [2] and changed its policies: Users are now allowed to communicate directly, but the new business model charges suppliers of used books at the time when they add the book to the system. It is very likely that less private customers will sell their books and one cannot assume that the seller has read the book. Thus, we think that there will be no community of interest that discusses the offered books.

customers, who were currently looking at the same books or CDs, 52% agreed and 38% said that they didn't want to be contacted.

The problematic issue is the notion of places in Odigo. Although many users are visiting the Amazon bookstore at each point in time, the probability that two users look at the same item synchronously is very low (about 1/2250 — the calculation will be explained in the second part of this paper). On the other hand, if all online users at Amazon.com would be displayed as possible interaction partners, this would lead to lists containing thousands of users, which is not very expressive.

3.3 Requirements

None of the above tools provides full support for communities of interest or communities of practice. Most of them lack in means for establishing the community (finding peers) and starting direct interaction.

A community e-commerce system should help the customers to get in contact based on their shared interests. It should also encourage the customers to keep the contacts alive. The buddy lists of Amazon show, how this can be achieved. Odigo supports partner finding of unknown users, but as discussed in the previous section, the selection of relevant users is not well supported.

As pointed out in the section on communities, the community needs social interaction to reach the stat of high social binding. Examples for social interaction are already realized in the system used by LandsEnd.com. Anyhow, the shared tools need to be integrated with contact facilitation technologies to allow new groups to form.

4 The GAMA-Mall Approach

To meet the requirements stated in the first part of this paper, we built the GAMA-Mall[3] system, which extends the Amazon.com bookstore. It is currently in an experimental stage, which allows to assess the technical feasibility of the system. The system will first be introduced following a usage scenario. After this, some technical details are examined: the system architecture, and the spatial structure of the GAMA-Mall, which allows detection of other customers who are present nearby.

4.1 Scenario

Imagine two users, Till and Bob, who are looking for books in the GAMA-Mall from their homes. To enter the store, Bob types the desired URL (in this case www.Amazon.com) in his browser. The system detects that Bob is not yet logged in to the system. Therefore, he gets redirected to a login page, where he can enter

[3] GAMA-Mall stands for a Mall based on the GAMA mechanism, that is Group Awareness based on Multiple Associations.

a personal nickname. He then continues browsing the pages of the bookstore, as if he was using a standard browser.

Till has also logged in to the store and started browsing books on programming. He has a specific interest on enhancing the quality of code and therefore navigates to books on refactoring. Bob didn't actually know which book he was looking for but he is also interested in programming. When he enters the page on eXtreme Programming, the system indicates that Till is browsing books with a related topic (cf. Figure 1) by adding user icons in front of the links that lead to the related book (this visualization was also used in the software development environment TUKAN [21]). The more red paint is used to draw the icon, the closer another user is. Colors range from dark red to green, which can be expressed as a mapping of the activity's intensity to the figure's hue.

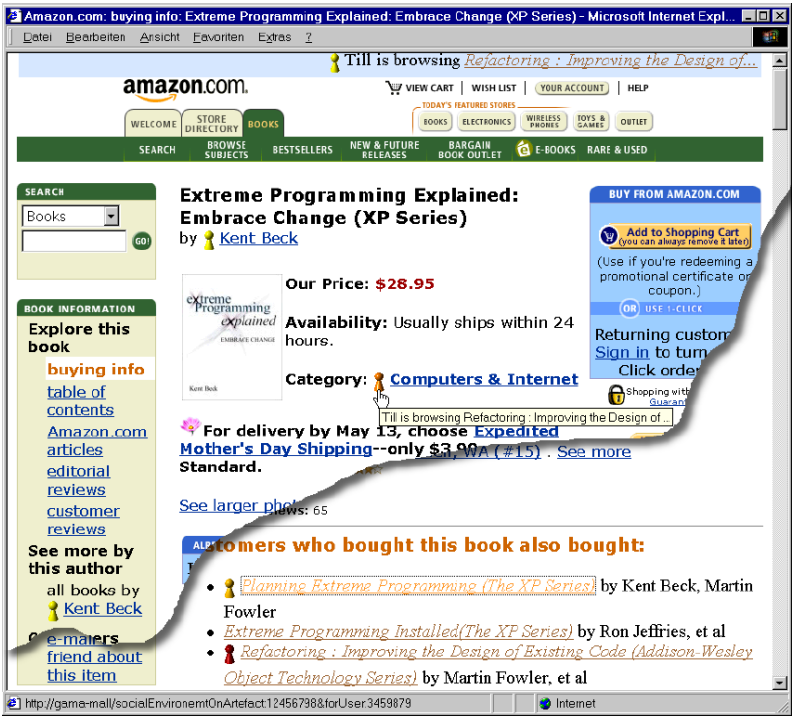


Fig. 1. Bob's view of the bookstore: Little indicators show him the presence of another user (Till), who is browsing related books. The color of the figures indicates the semantic distance to Till following the link next to the figure. If the figure tends to be greener, then it connects to Till on a long path. A dark red figure indicates a direct connection to Till. Two screenshots were combined for space reasons.

Using the context information that is attached to the figure, Bob can decide if it is worth following the link to Till. He can also open the collaboration inspector (not shown), that informs him about Till's present activities and possible tools for tighter collaboration. Bob decides that Till's topic is interesting for him and starts a chat session, where Till and Bob discuss the value of redesign in software development. During the discussion, they notice that their interests are close and decide to continue shopping in a coupled way. Till activates the browser coupling and from that time, they navigate through the pages together. Whenever Till or Bob presses a link, the partner's browser is also redirected to the new page.

4.2 System Architecture

The GAMA-Mall is implemented with the architecture shown in Figure 2. The central part of it is the proxy server. All requests for pages of the web store from the client are directed through the proxy. The proxy then retrieves the desired page from the web and adds two additional parts to it, before delivering it to the client: The awareness info and an applet, which enables the client's web browser to listen for update events.

To calculate the awareness info, the proxy has to have access to the shared collaborative object space, which is provided by the collaboration server. Therefore, the proxy is also a COAST client and the collaboration server is a COAST server (cf. [19] for a description of the COAST infrastructure).

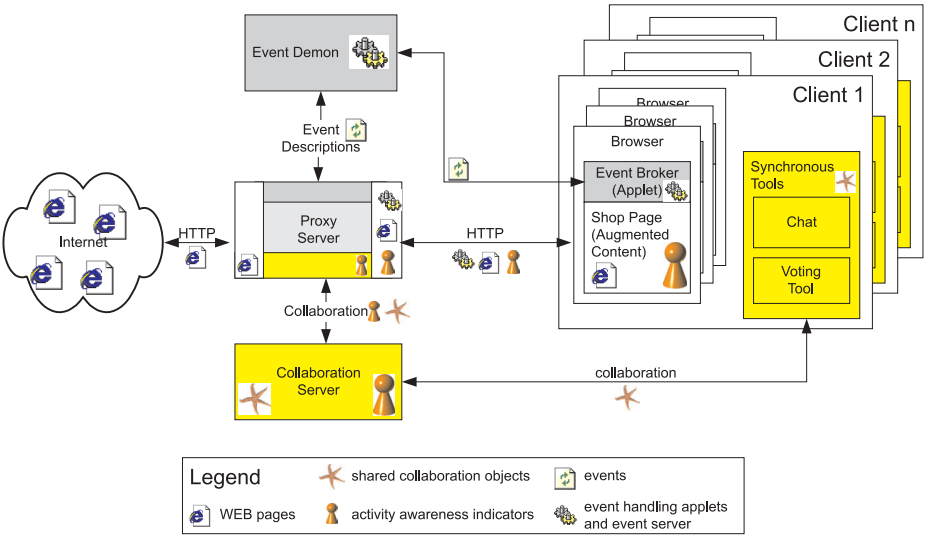


Fig. 2. The architecture used for the GAMA-Mall.

The same collaboration server is accessed by the clients, if they want to enter tighter collaboration. Whenever they request a synchronous tool, the proxy server detects, if the client has already started a COAST client. If this is true, the proxy just creates a shared application model for the client and the COAST infrastructure will ensure that the client receives a new synchronous application model at his machine (cf. [20] for an explanation of shared application models in COAST).

The second information channel between the proxy and the browser is modeled using the event demon and the event broker. They will be covered by the next section.

4.3 Automatic Updating Browser Content

Normally, web pages are only received by clients. Whenever a client is interested in a page, it requests the page from the server and the server answers the page. This model is called the pull model, since the client has to pull the information from the server. Updates can only be requested by the client. Thus, if the content changes, the client will still display outdated information. The client could request the page frequently (polling), but this will result in a very high network traffic (cf. [11]).

Another way to transfer information is known as push technology. In this case, the server transmits new information to all interested (subscribing) clients, without the necessity of a specific client request before delivery ([8]).

This approach ensures that only new information is broadcast, but it is only suitable, if the new information is equal for all clients. If the Information differs (e.g. in the case of personalizes shop pages), this approach would result in a complex connection handling procedure within the proxy server.

We therefore use a third approach, which combines push and pull technology: the push-triggered-pull approach ([10]). In this approach, the proxy broadcasts only update events to the clients and the clients may then decide to request for the reload of a page, after receiving the update event.

Besides the update event, the proxy can also generate load events. These events are used to force other clients to move to a new page in the case of collaborative browsing. Whenever the guiding user requests a new page, the proxy will notice this and send a load event, which contains the URL of the new page to the other members of the cooperative browsing group.

4.4 Spatial Representation

When users browse through the virtual bookstore, they get informed, who is nearby. A simple implementation was provided by the Odigo-System, which distinguishes three different degrees of nearness: Two users can be on the same page, they can be on the same site, or they can be at different sites.

The GAMA-Mall has a more complex notion of the space. Each artifact of the store (books, categories, or authors) is analyzed and related to the artifacts

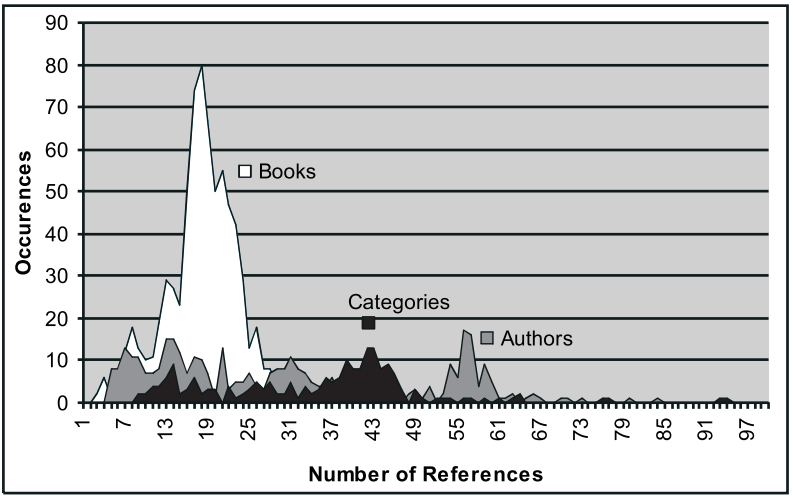


Fig. 3. Complexity of connections between artifacts. Sample taken from 1900 artifacts.

that can be reached from the source artifact. The analysis is done incrementally: Whenever a customer is interested in a page (describing an author, a book, or a category), the GAMA-Mall-System analyzes the page's content and searches links to other artifacts. The links are attached with weights based on the number of links that lead from the source artifact to the target. If two artifacts are related by many links, then the distance between these two artifacts is very small. Using this simple analysis technique leads to good results regarding the semantic structure. The Author of a book is for instance always very close to the book that he wrote. He is not so close to the book, if he is just an author, who is read by the same customers, as the first author.

To get concrete numbers of the graph's complexity, we parsed 1900 artifacts. In average, each book links to 18.28, each author to 29.72, and each category to 34.52 artifacts. The overall average number of edges is 27.5. Figure 3 shows the distribution curve of artifacts according to their number of relations to other artifacts.

The number of related artifacts directly influences the calculation of near artifacts. Two artifacts are near to each other if there is a short path between these artifacts. The larger the number of relations is, the larger is the number of nodes found in the surrounding area. If the number of users that are present on the web site at the same time and the number of artifacts on the site is known, then the nimbus radius can be calculated according to the desired probability of meeting another user.

From internet usage statistics, we know that in average 12.000 users are visiting Amazon at each second (calculated from [14] using the numbers from March 2001 to February 2002). Press releases from Amazon [3] reveal that the

shop holds more than 28 Mio. items. If users should meet with a probability of 50%, then the nimbus should contain 1150 artifacts, which leads to a needed nimbus radius r_n of 3.64. Taking in account that there are peak times, where many users visit the site (and that not all items are visited frequently), the probability would even be higher when using the calculated nimbus radius.

5 Related Work

Most of the related work was already mentioned in Section 3. As we pointed out, each of the solutions has drawbacks concerning the process of finding other online customers and tight synchronous interaction.

Some approaches influenced this work from a technical point of view: The WBI architecture proposed by Barrett and Maglio [4] uses proxies to augment web pages according to the user's needs. The use of proxies for collaborative browsing was presented by Cabri et al. [5]. The approach also uses applets that are included by the proxy to couple and synchronize the web browsers. Our approach differs in the fact that we augment the page's content with awareness information to enable contacts. The CoBrow [22] system shows, what other users are currently browsing the same page or a page in a logical vicinity. This approach is very close to the GAMA-Mall approach, but again, the augmentation of the pages is not done in CoBrow and thus there is no real integration of users within the shop. Finally the CoCoBrowse System [25] aims to make use of existing standards for presence services [9,23] to track the users' positions. These standards ease the spreading of the system since less components are required. On the other hand, these standards also restrict the possible information. A dynamic analysis of spatial structures is not yet part of CoCoBrowse. Integrating the ideas of standard based presence information with the GAMA-Mall-Approach could be a challenging task for future work.

6 Conclusions

In this paper we analyzed existing real and virtual shopping solutions and whether they support communities. We focused on communities of interest and communities of practice, since these communities promise to be most valuable for communities of customers. The analysis revealed that current community support in electronic commerce can be extended to reach communities of practice or communities of interest. The missing parts are contact facilitation and support for collaborative synchronous interaction.

The GAMA-Mall system aims to fill this gap by providing awareness on nearby customers based on a spatial arrangement of the shop's artifacts. Collaborative tools are integrated in the GAMA-Mall architecture to enhance better social involvement. As the statistical findings of the prototype show, the size of the user's nimbus (which defines, how far nearby customers are detected) can be adjusted to match a desired probability of signaling contacts.

The system is currently in a state that first functional tests can be run. However, to perform a usability study, the problem of the critical mass has to be solved. The best way would be, to integrate the system in the real Amazon.com store. But, since this is running business application, this will not be possible at the current state. Therefore, we aim on performing small lab studies with users having comparable tasks to solve, to see, whether or not customers meet in the store and communication and collaboration takes place.

Regarding related work, it would be an interesting next step to combine the GAMA-Mall architecture with presence protocol standards. As demonstrated by the CoCoWare approach, this could ease the distribution of the system and speed up the process of reaching a critical mass.

Acknowledgements. This work was partially funded by the PhD program *Enabling Infrastructures for Electronic Commerce* at the Technical University of Darmstadt. I would like to thank my former colleagues at Fraunhofer IPSI, especially Torsten Holmer for supporting the user studies with Odigo and Alejandro Fernandez, who contributed to very fruitful discussions on communities. Ludger Fiege and Gero Mühl developed the REBECCA system, which was used to propagate events. Thanks are also due to the OpenCoast development team who developed the COAST framework (especially Christian Schuckmann and Jan Schümmer). Finally, I would like to thank my advisor Jörg Haake (FernUni Hagen) for counseling my research.

References

1. AOL: "AOL Instant Messenger", `http://aim.aol.com/`, 2001.
2. Advanced Book Exchange Inc.: "Abebooks.com acquires Europe's JustBooks", `http://dogbert.abebooks.com/abe/TextToHtml?t=Media+Room&h=x&f=abemedia/pressbox/2001/03102001.htm`, 2001.
3. Amazon.com: "Amazon.com Introduces Worldwide Digital Group", `http://www.iredge.com/iredge/iredge.asp?c=002239&f=2005&fn=Digital3_8_01__337.htm`, 2001.
4. Barrett, R.; Maglio, P.: "Intermediaries: New places for producing and manipulating web content", 7th International WWW Conference, (Computer Networks, Band 30Elsevier: Brisbane, 1998, 509–518.
5. Cabri, G.; Leonardi, L.; Zambonelli, F.: "Supporting Cooperative WWW Browsing: a Proxy-based Approach", 7th Euromicro Workshop on Parallel and Distributed Processing, Madeira, Portugal, 1999, 138–145.
6. Carotenuto, L.; Etienne, W.; Fontaine, M.; Friedman, J.; Muller, M.; Newberg, H.; Simpson, M.; Slusher, J.; Stevenson, K.: "CommunitySpace: Toward Flexible Support for Voluntary Knowledge Communities", online proceedings of "CHANGING PLACES - the workshop on workspace models for collaboration", London, 1999. `http://www.dcs.qmw.ac.uk/research/distrib/Mushroom/workshop/final-papers/lotus.pdf`.
7. Chapman, R.: "Community continuum – Building online communities on the Web", `http://www.informationhighways.net/mag/mprevious/00apr01.html`, 2001.

8. Datta, A.; Celik, A.; Biliris, A.; Wright, R.: "SubScribe: Secure and Efficient Data Delivery/Access Services in a Push-Based Environment", Proceedings of the First International Conference on Telecommunications and Electronic Commerce (ICTEC), Nashville, TN, 1998.
9. Day, M.; Rosenberg, J.; Sugano, H.: "A Model for Presence and Instant Messaging", 2000.
10. Fiege, L.; Mühl, G.; Gärtner, F.: "A Modular Approach to Build Structured Event-based Systems", ACM Symposium on Applied Computing (SAC) 2002, to be published, 2002.
`http://www.gkec.informatik.tu-darmstadt.de/fiege/sac02.pdf`.
11. Franklin, M. J.; Zdonik, S. B.: "Data In Your Face: Push Technology in Perspective", in Haas, L. M.; Tiwary, A., Ed., SIGMOD 1998, Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM Press: Seattle, Washington, USA, 1998, 516–519.
12. Hardach, G.; Schilling, J.: "Das Buch vom Markt", Luzern, Frankfurt/M: Bucher, 1980.
13. ICQ: "Company Homepage", `http://www.icq.com/`, 2001.
14. INT Media Group: "Cyberatlas: Traffic Patterns", `http://cyberatlas.internet.com/big_picture/traffic_patterns`, 2002.
15. Linden, G. D.; Jacobi, J. A.; Benson, E. A.: "Collaborative recommendations using item-to-item mappings", United States Patent 6266649, 2001.
16. Marathe, J.: "Creating Community Online", Durlacher Research Ltd, 1999.
17. Odigo: "Company Homepage", `http://corp.odigo.com/`, 2001.
18. Preece, J.: "Online Communities", Chichester, UK: Wiley, 2000.
19. Schuckmann, C.; Kirchner, L.; Schümmer, J.; Haake, J. M.: "Designing object-oriented synchronous groupware with COAST", Proceedings of ACM CSCW'96 Conference on Supported Cooperative Work, Boston, Mass., 1996, .
20. Schuckmann, C.; Schümmer, J.; Seitz, P.: "Modelling collaboration using shared objects", Proc. of the Conference on Supporting Group Work, 1999, 189–198.
21. Schümmer, T.: "Lost and Found in Software Space", Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34), Collaboration Systems and Technology, IEEE-Press: Maui, HI, 2001, .
22. Sidler, G.; Scott, A.; Wolf, H.: "Collaborative Browsing in the World Wide Web", Proceedings of the 8th Joint European Networking Conference (JENC8), Edingburgh, 1997, 122 (122–1 to 122–8).
23. Sparks, R.; Peterson, J.: "SIP Extensions for Presence", 2001.
24. Telewest: "Online communities can keep your custommers coming back for more", `http://www.telewest.co.uk/yourbusiness/newsandinformationtechnology10.html`, 2001.
25. Ter Hofte, G. H.: "Place-based presence as a Basis for ad-hoc Communication", Bonn, Germany: 2001.
26. Warms, A.; Cothrel, J.; Underberg, T.: "Return on Community: Proving the Value of Online Communities in Business", Participate.com, 2000.

Modelling Service-Providing Location-Based E-communities and the Impact of User Mobility

Seng Wai Loke

School of Computer Science and Information Technology
RMIT University
GPO Box 2476V
Melbourne VIC 3001, Australia
swloke@cs.rmit.edu.au

Abstract. As short range wireless networks (such as wireless local area networks (WLANs) and Bluetooth) become widespread, an infrastructure is emerging on which we envision advanced value-added services. Such an infrastructure can provide access to not only services on the greater Web but also services which are locally relevant and more tailored to the user's current context (e.g., the user's location, the function of a place the user is currently at, other people who are also in the same network at that time, etc). Such short range wireless networks are convenient technology for the creation of *location-based e-communities* or *LE-communities* which are counterparts of real-world location spaces. Each LE-community provides services to users and users can participate in many LE-communities. Hence, there would be a proliferation of services to a user available via numerous LE-communities, and these services would continually change as the user moves physically into and out of the range of the wireless networks (and so, into and out of their corresponding LE-communities). In this paper, we study the impact of user mobility on services provided by LE-communities. Our contribution here is a model to enable reasoning about which services can be invoked as the user moves, and present operators for combining the services offered by different LE-communities inspired by operators for combining logic programs. Our model is a first step towards a programming model for applications involving LE-communities.

1 Introduction

While an increasing percentage of the world's population is getting connected (almost) anytime anywhere via the mobile cellular network, there has been growing excitement over short-range wireless networking technologies [5]. Examples of these technologies include wireless local area networks (WLANs) (e.g., the IEEE 802.11 LANs with access points having a limited range of roughly a hundred metres) and wireless personal area networks (WPANs) (e.g., Bluetooth[1] with a limited range of roughly 10 metres).

[1] See http://www.bluetooth.com

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 266–277, 2002.

802.11 networks are becoming ubiquitous and are starting to appear in homes, offices and public places such as shopping complexes, airports, hotels, and restaurants. Internet Service Providers (ISPs) are also appearing that provide public access points to allow wireless access for subscribers.[2] WLAN services are finding their way into eight Florida shopping centers in the US.[3] This reduces the need for extensive wiring to provide various applications to merchants such as remote store monitoring using wireless cameras and devices that count the number of customers. The merchants have yet to extend wireless services to customers. The Bryant Park Hotel in New York lets its staff service guests using information provided via wirelessly networked devices.[4] Starbucks cafe is providing WLAN services to customers, but so far these services are of a horizontal (general) nature (Web, email, etc).[5] The WLAN company MobileStar hopes to implement 8000 WLAN sites in the US by the end of 2003. Europe has already installed several hundred public WLAN access points.[6] Moreover, Bluetooth public access points are emerging as well. Ericsson's Bluetooth Local Infotainment Point (BLIP) devices provide Bluetooth access points to information for Bluetooth enabled PDAs and cellphones, and is undergoing public trials in Japan [3].

As these short range wireless networks become widespread, an infrastructure is emerging on which we envision advanced value-added services. Such an infrastructure can provide access to not only services on the greater Web but also services which are locally relevant and more tailored to the user's current context (e.g., the user's location, the function of a place the user is currently at, other people who are also in the same network at that time, etc). Such services can be of varying levels of specialization and might be formed by extracting Web services relevant to the user's context. A tourist in a room within a museum might access (via his/her mobile device) an information service that provides descriptions about articles in the room as well as access services relevant to the entire city (being still a tourist, albeit a museum visitor).

Technologies are also being developed that allow moving between network connections. For example, one could connect to a WLAN while within range of its access point, then hand-off to a lower speed wireless telecommunications service once out of range, and then reconnect to another access point in a different WLAN.[7]

The above developments form a basis for the idea of electronic communities, or *e-communities* for short, that are induced by a location and the (possibly temporary) geographical collocation of people. The community carries the idea of richer member participation and interaction. These communities can build on wireless networking technologies [6]. For example, the e-community of a hotel is supported by a WLAN for hotel guests and hotel management. Specific

[2] See `http://www.80211-planet.com/news/article/0,4000,1481_859901,00.html`
[3] See `http://www.luent.com/micro/NEWS/PRESS2001/080801a.html`
[4] See `http://www.mobileinfo.com/News_2001/Issue34/Symbol_Bryant.htm`
[5] See `http://mcommercetimes.com/Services/155`
[6] `http://www.mobileinfo.com/News_2001/Issue39/Frost_WLAN.htm`
[7] iConverse provides this. See `http://www.useit.com/alertbox/20010916.html`

applications are build on top of the WLAN infrastructure, which can be used to provide services (the service provides a structuring abstraction for applications), and then these services are structured into e-communities. Note that not all such WLANs will develop fully into active e-communities - one might stop at providing services to individual users, without users necessarily interacting among themselves (e.g., each user can do Web browsing but is unaware of other people connected to the same WLAN).

Short range wireless networking are convenient technology for the creation of virtual spaces which are counterparts of real-world location spaces, particularly for users on the move. An individual in a hotel has not only the hotel as a physical community but also a corresponding e-community supported by the WLAN, which we call a *location-based e-community* or *LE-community*. The individual is able to access services or interact with other individuals in the physical (and virtual) community. A LE-community is thus an e-community that is induced by (and is a counterpart of) an individual's surroundings or location. We see LE-community as, roughly, the highest development in a layered model shown in Figure 1.

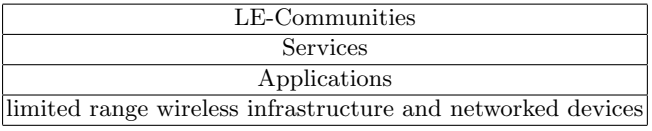


Fig. 1. Layers of development towards LE-communities.

Note that the individual might still have access to virtual communities on the Web (which are geographically transcending and location-independent), and might only be at a location for a short time (and so, have only minimal or rapid interaction with a LE-community).

An individual can be in many LE-communities at the same time. For example, one could be in a cafe in a shopping complex in some town, and so, could then be in three LE-communities at the same time: the cafe LE-community, the shopping complex LE-community, and the town LE-community. An individual can move out of one cafe (and so out of that cafe's LE-community) into another cafe (and so into this cafe's LE-community) while still being in some other LE-community (-ies). Also, assuming each LE-community provides a set of services to its members, an individual should therefore be able to access some combination of the services offered by the LE-communities in which the individual currently dwells. This combination of services might take a number of different forms. For example, similar or same services in different LE-communities might be unioned, the services in an inner LE-community might have precedence over similar services in an outer LE-community, or only services available in both the inner and outer LE-communities can be used by the individual.

It would be useful to be able to reason more precisely about the above scenarios (discussed informally above), and to develop a programming model for applications involving LE-communities.

In this paper, we attempt to develop a formal model of the above scenarios in terms of a system of transition rules. We first provide a more precise notion of LE-communities and relationships between LE-communities which captures the impact of an individual's movement on his/her participation in LE-communities. The ideas are partially inspired by the ambient calculus [2]. Then, we present operators for combining the services offered by different LE-communities inspired by operators for combining logic programs developed in [1]. In the following section, we describe notation for LE-communities, and membership in LE-communities. Then, in §3, we present operators for combining services of LE-communities. In §4, we discuss the interplay of the user's movement with compositions of services. We conclude in §5 with future work.

2 LE-Communities

This section presents notation to represent the LE-communities (which we abbreviate to *LEC*) an individual (which we simply call the *user* (of some mobile device and of services of LECs)) can be a member of (and participate in).

Let $\mathcal{L}$ be a finite set of LECs (or the set of symbols representing the LECs). We shall often use letters A, B, C,...$\in \mathcal{L}$ to denote LECs. At any time, a user can be in several LECs at the same time. Given a finite set of users $\mathcal{U}$, we also define *users' configuration* denoted by Σ that is a function that maps each user to a set of LECs (representing the LECs that the user is in) (i.e., $\Sigma \subseteq \{(u, \Sigma(u)) \mid u \in \mathcal{U}, \Sigma(u) \subseteq \mathcal{L}\}$).

A user u's exit or entry operations affect Σ. We can model these operations as functions that map from one users' configuration to another:

1. exit a LEC:
$$\Sigma \xrightarrow{u \ exit \ A} \Sigma'$$
where $\Sigma' = \Sigma \setminus \{(u, \Sigma(u))\} \cup \{(u, \Sigma(u) \setminus \{A\})\}$. Σ' differs from Σ if $A \in \Sigma(u)$. If $A \notin \Sigma(u)$, then the *exit* A operation effectively does nothing.
2. enter a LEC:
$$\Sigma \xrightarrow{u \ enter \ A} \Sigma'$$
where $\Sigma' = \Sigma \setminus \{(u, \Sigma(u))\} \cup \{(u, \Sigma(u) \cup \{A\})\}$. Σ' differs from Σ if $A \notin \Sigma(u)$. If $A \in \Sigma(u)$, then the *enter* A operation effectively does nothing.

These two operations can be performed by the user in two ways: explicitly – when the user explicitly chooses to exit or enter a LEC via some interface (e.g., an option to enter a LEC is presented to the user on his/her mobile device when the user comes within range of the LEC's WLAN), or implicitly – when the user physically moves in or out of the range of a LEC (e.g., when the user moves out of the range of a WLAN). The physical movements of the user and

the exit/entry selections of the user over time effectively yield a sequence of operations that establishes a sequence of configurations starting from an initial users' configuration. Similarly, another user can also establish such a sequence. How these sequences combine depends on the timing of the operations. So, given a set of users, at a given point in time, there is a users' configuration Σ that is a snapshot of the users in LECs - the set of users in one of the LECs A is then $\{u|A \in \Sigma(u)\}$.

3 Operators for Composing Services of LE-Communities

So far, we have taken an opaque picture of LECs. We now consider each LEC A as having a finite set of services Ω_A, where services can be invoked by a user. More complex definitions of services can be employed, but to provide a concrete example, we can simply view a service as a method (e.g., a method in a programming language such as Java or a CORBA component, or a Web service [4], and Ω_A is then implemented as an object or a component), and a request to invoke a service as a method call. We shall assume there is a mechanism to match service invocation requests to services (in our example, it is the mechanism that matches a method call with a method definition). The discussion that follows is independent of the specifics of the matching mechanism and we only assume that this matching mechanism is a (boolean) function *match* that takes a service request r (in some universe of requests) and a service s (in some universe of service descriptions) and returns either true or false.

At a given point in time, a user u is often in several LECs but not all the services of all these LECs are equally available to the user, due to constraints imposed by the LECs, intermediaries between LEC services and users (analogous to application service providers which package and manage applications for users and act as intermediaries between users and application originators, we might have LEC service providers that manage LEC services for users), or the users themselves. Hence, we contend that it is not sufficient, in general, to say that the services available to the user are all the services in $\Sigma(u)$, but more precisely, the services available to the user is some composition of the services in $\Sigma(u)$.

We define operators that compose the services offered by LECs in terms of rules that describe what it means for a service to match against a composition. The rules makes precise several intuitions about how a user requests services from a combination of LECs.

Given a composition of services $\mathfrak{C}$ and a request r to invoke a service, we write $\mathfrak{C} \Vdash r, \Theta$ to mean r successfully matches against $\mathfrak{C}$, returning a set of matching services to invoke (which we call the *invocation set*, denoted by Θ). For some LEC A, the statement $\Omega_A \Vdash r, \Theta$ means that r matches some service s in Ω_A, and Θ contains that service - if there are several matching services, Θ contains only one of them (and there might be a mechanism to select which one but we do not prescribe that here):

$$\Omega_A \Vdash r, \Theta \quad \text{iff} \quad \Theta = \{s\}, \text{ where } s \in \Omega_A \text{ such that } match(r, s) = true$$

If r does not match any service in Ω_A, then the statement $\Omega_A \Vdash r$ does not hold, denoted by $\Omega_A \not\Vdash r$. Note that the relation $\Vdash$ focuses only on the matching, and not on the results of service invocation: r might have been invoked and some computation triggered, but the invocation might not return desired results. We do not define or deal with results of service invocations but only whether a request is matched.

We provide rules below that effectively define the relation $\Vdash$ between compositions and requests. The rules are of the form:

$$\frac{premises}{conclusion}$$

which means *conclusion* holds whenever *premises* hold.

Compositions are expressions formed using the operators described below. These operators form expressions $\mathfrak{C}$ whose syntax is given in EBNF as follows:

$$\mathfrak{C} \ ::= \ \Omega_{\mathbb{L}} \mid \mathfrak{C} \cup \mathfrak{C} \mid \mathfrak{C} \cap \mathfrak{C} \mid \mathfrak{C} | \Omega_{\mathbb{L}} \mid \mathfrak{C} \triangleleft \Omega_{\mathbb{L}}$$

where $\mathbb{L} \ ::= \ A \mid B \mid C \mid \ldots (\in \mathcal{L})$.

The meaning of the operators are as follows:

- union (denoted by "$\cup$" - relying on context to differentiate from set-theoretic union): Given compositions $\mathfrak{C}$ and $\mathfrak{D}$, we have

$$\frac{\mathfrak{C} \Vdash r, \Theta}{(\mathfrak{C} \cup \mathfrak{D}) \Vdash r, \Theta} \qquad \frac{\mathfrak{D} \Vdash r, \Theta'}{(\mathfrak{C} \cup \mathfrak{D}) \Vdash r, \Theta'}$$

  The above rules state that a service r matches against the composition $\mathfrak{C} \cup \mathfrak{D}$ if either r matches against $\mathfrak{C}$ or against $\mathfrak{D}$ (and it does not matter which). To a user, such a union means services in both compositions are available. There is non-determinism inherent with these rules and either the invocation set Θ or Θ' is used.
- intersection (denoted by "$\cap$"): The rule is

$$\frac{\mathfrak{C} \Vdash r, \Theta \quad and \quad \mathfrak{D} \Vdash r, \Theta'}{(\mathfrak{C} \cap \mathfrak{D}) \Vdash r, (\Theta \cup \Theta')}$$

  The rule means that some services Θ in $\mathfrak{C}$ must have matched with r and some services Θ' in $\mathfrak{D}$ must have matched with r. Note that, if $\mathfrak{C}$ is Ω_A and $\mathfrak{D}$ is Ω_B this implies a relationship between A and B, that they each have a service that matches the given request. In other words, intersection relies on "something in common" between the LECs in order to work. To a user, a request to an intersection is a means of testing the similarity between the LECs as well as seeking responses from different communities on the same request (e.g. to check agreement between results of similar services). Note that the set-theoretic union of the invocation sets are used $\Theta \cup \Theta'$, that is, services from both $\mathfrak{C}$ and $\mathfrak{D}$ will be invoked.
- restriction (denoted by "$|$"): This operator expresses how the services of one LEC can suppress that of others:

$$\frac{\mathfrak{C} \Vdash r, \Theta \quad and \quad \Omega_A \not\Vdash r}{(\mathfrak{C} \mid \Omega_A) \Vdash r, \Theta}$$

Recall that $\Omega_A \not\Vdash r$ holds if and only if we cannot find a matching service for r in Ω_A. So, only requests that that do not match any service of A can be matched with restriction. It is in this sense that the services in A restricts the services in $\mathfrak{C}$. This models the case of how some of the services in one LEC is rendered ineffective when the user enters another LEC. This could model a filtering effect of LEC membership, and also provides a basis for defining overriding below.

– overriding (denoted by "$\triangleleft$"):

$$\frac{(\mathfrak{C} \mid \Omega_A) \Vdash r, \Theta}{(\mathfrak{C} \triangleleft \Omega_A) \Vdash r, \Theta} \qquad \frac{\Omega_A \Vdash r, \Theta'}{(\mathfrak{C} \triangleleft \Omega_A) \Vdash r, \Theta'}$$

In fact, by definition of "$\cup$" above,

$$\mathfrak{C} \triangleleft \Omega_A = (\mathfrak{C} \mid \Omega_A) \cup \Omega_A$$

The rules state that either a service of A is used or a service of $\mathfrak{C}$ which is not available in A is used. So, if r matches a service of $\mathfrak{C}$ and also a service of A, then the service of A will be invoked. This means that the services of A overrides that of $\mathfrak{C}$. This operator can be used to model how the services of a more current (or structurally, an inner LEC) takes precedence over those of an earlier (or outer) LEC. For example, the services offered by a cafe in a shopping complex might override similar services offered by the shopping complex.

So, given a request and a composition such as $((\Omega_A \cup \Omega_B) \cap \Omega_C) \Vdash r$, the above rules effectively computes an invocation set.

4 The Impact of User Mobility on Services

We next explore the interplay between a user's movement and compositions of services, and also briefly explore several implementation issues.

4.1 Composition Schemes

At any point in time, a user u might have configuration $\Sigma(u)$, and the user can only make use of services in the LECs that it is a member of, i.e. the LECs in $\Sigma(u)$. A request r of the user will therefore be made against a composition of LECs in $\Sigma(u)$. For example, if $\Sigma(u) = \{A, B, C\}$, and these LECs are composed in the manner $(\Omega_A \cup \Omega_B) \cap \Omega_C$, then processing r will evaluate the statement $((\Omega_A \cup \Omega_B) \cap \Omega_C) \Vdash r$. There is a question of how this composition is formed at any given point in time, and this question is answered via a *composition scheme*.

A composition scheme is needed as $\Sigma(u)$ changes each time a user exits or enters a LEC, and the composition to be used might depend on the request. Two possible composition schemes are:

1. The composition to form is specified by the user when r is issued. This means that r can be issued against any user-specified composition $\mathfrak{C}$ as long as the LECs mentioned in $\mathfrak{C}$ are in $\Sigma(u)$.
2. Given $\Sigma(u)$ at a point in time, there can be a fixed manner (a system default meta-composition rule that is user or system-specified) in which a composition is formed. For instance, regardless of what is in $\Sigma(u)$ at a given point in time, a default rule could be that, all requests are processed against the union of all LECs in $\Sigma(u)$, i.e. $(\bigcup_{X \in \Sigma(u)} \Omega_X) \Vdash r$.
   Incremental rules that map from one composition to another are possible, that specify how a given LEC should be composed with the existing composition $\mathfrak{C}$, whenever an entry or exit operation occurs. For example, such an incremental entry rule might take the form:

$$\mathfrak{C} \xrightarrow{u \ enter \ A} \mathfrak{C} \oplus \Omega_A$$

   where $\oplus$ is one of the above operators. A commonly used composition scheme might be the case where $\oplus$ is $\triangleleft$, so that from a history of entries, we obtain expressions of the form $\Omega_A \triangleleft \Omega_B \triangleleft \Omega_C$ which always gives the inner (later) LEC precedence over outer (earlier) LECs.
   Also, for exit, we can define the following incremental exit rule:

$$\mathfrak{C} \xrightarrow{u \ exit \ A} \mathfrak{C}'$$

   where $\mathfrak{C}'$ is an expression formed from $\mathfrak{C}$ by deleting all occurrences of Ω_A from $\mathfrak{C}$. For example, $((\Omega_A \cup \Omega_B) \cap (\Omega_C \mid \Omega_A))$ becomes $\Omega_B \cap \Omega_C$.

The first scheme provides the greatest flexibility and control to the user, but may be cumbersome for someone on the move. The second scheme is easier for the user, since there is a well-defined default behaviour, and a well-defined effect of the user's movement on the composition.

Two types of user requests can be defined, one which is persistent (also often called continual queries) and so is automatically invoked after each exit and entry operation, and ones which are once-off, i.e. invoked once against a specified composition and not automatically reinvoked after a change in the user's LECs.

4.2 An Example Scenario

To illustrate our model, we consider a scenario of a user who is touring a town T, staying in hotel H, and attending a conference at site C in H. The person is also a member of a company X, is currently sitting in a cafe A in a shopping complex S, away from the site C. The LECs are T, H, C, X, A, and S. Let us assume the following current composition of services for the user:

$$\Omega_X \cup ((\Omega_T \triangleleft \Omega_S) \triangleleft \Omega_A)$$

This expression states that the services of A would override those of S, and those of S would, in turn, override the similar services of T, but the services of X are

always available. For example, consider a request for music services (e.g., MP3 selection and download) which would take the form:

$$(\Omega_X \cup ((\Omega_T \triangleleft \Omega_S) \triangleleft \Omega_A)) \Vdash music$$

The rules given in §3 can be used to select the music service to invoke.

Now, consider the following movements of the user:
exit A
exit S
enter H
enter C
Note that we only consider here how the user's movement relates to A, S, H, and C - the user might enter or exit different LECs between the shopping complex and the hotel. The movements above effectively creates a simple "program" which impacts on the services available to the user. Using the incremental exit rule from the previous section, the new composition after *exit A* is then:

$$\Omega_X \cup (\Omega_T \triangleleft \Omega_S)$$

And after *exit S*, it is

$$\Omega_X \cup \Omega_T$$

Thereafter, using the following incremental entry rule:

$$\Omega_X \cup \mathfrak{C} \xrightarrow{u\ enter\ A} \Omega_X \cup (\mathfrak{C} \triangleleft \Omega_A)$$

the operation *enter H* gives:

$$\Omega_X \cup (\Omega_T \triangleleft \Omega_H)$$

and *enter C* gives:

$$\Omega_X \cup ((\Omega_T \triangleleft \Omega_H) \triangleleft \Omega_C)$$

What we have achieved via this modelling is that we know more precisely what (combination of) services are available to a user while in some configuration. Given a composition scheme, such modelling can also be used to predict the impact of user's movements on services without actual movements. We can also reason about persistent requests. Consider a persistent request for music services (e.g., MP3 selection and download) issued in the initial composition, which would take the form:

$$(\Omega_X \cup ((\Omega_T \triangleleft \Omega_S) \triangleleft \Omega_A)) \Vdash music$$

Subsequent movements would have the *music* request evaluated against different compositions:

$$\begin{array}{rl}
(\text{after } exit\ A) & (\Omega_X \cup (\Omega_T \triangleleft \Omega_S)) \Vdash music \\
(\text{after } exit\ S) & (\Omega_X \cup \Omega_T) \Vdash music \\
(\text{after } enter\ H) & (\Omega_X \cup (\Omega_T \triangleleft \Omega_H)) \Vdash music \\
(\text{after } enter\ C) & (\Omega_X \cup ((\Omega_T \triangleleft \Omega_H) \triangleleft \Omega_C)) \Vdash music
\end{array}$$

We could then see which music services in which LEC would be invoked by the *music* request using the transition rules in §3.

So far, we have not yet distinguished push and pull services. The requests described in previous sections might be requests for push services such as wireless advertising. For wireless advertising, the overriding operator provides a means of restricting advertisements to those more locally relevant. For example, when a user walks along a street lined with BLIP points, the user might remain in a greater LEC but enters and exits the LECs of BLIP points as he or she walks (and could simultaneously be in the LECs of several BLIPs at the same time). The user would then be using the advertising services of different BLIPs while moving.

As an example of the use of $\cap$, the user might also want to test and compare similar services from different LECs (e.g., to check agreement), say to check tourist services from the town and the conference site, and if authorized, can issue requests against a composition such as:

$$\Omega_T \cap \Omega_C$$

4.3 Implementation Issues

For scenarios such as the above, we envision the following implementation. In a LEC service provider model, we can allow the LEC service provider to keep track of which LECs a user is in as well as the current composition. Alternatively, the mobile device might keep track of a user's LECs and the current composition. The detection of user entry and exit can be done by the networking technology as soon as a user moves into or out of range and this information is provided to the mobile device or LEC service provider. When the user issues a service request via his or her mobile device, the request might be handled entirely on the mobile device if service code has been transferred into the user's device or the request can be transferred over the wireless network via access points to servers implementing LEC services.

The user can make use of Web or WAP (depending on the device at hand) forms to issue requests. Appropriate LEC pages can be pushed to the user when he or she enters a LEC. A LEC can have a Web site (hosting a virtual community about the LEC) accessible from anywhere where users in their own time, can register his/her devices and preferences with the LEC, before the user actually visits the LEC.

A question to address is how to decide what combination of services is available for a request of a given user at a given time. We can discuss this question in the context of three parties that we think should have a say in this decision: the user, a LEC, and the LEC service provider. A LEC would have its own preference about how its services should be used (e.g., that its services should override that of any others while the user is in that LEC). A user would have its own preferences (e.g., to receive advertising from all its LECs rather than just from some). A LEC service provider (with agreement from LECs) might like to offer its own mix of services to the user at any given point in time (e.g., a LEC

service provider might have an agreement with a LEC that its services should not be overriden while the user is in the LEC). Conflicts among the parties on such preferences could arise.

One solution to this question is to give the user full control. There would be a pre-programmed default composition scheme which the user creates (or chooses from a selection offered by a LEC service provider). In addition, the (more diligent) user, who must then learn the operators and acquire knowledge about what services are available, can specify a different composition for each request. Another solution is to employ real-time negotiation for services on entry into a LEC. We will continue to explore other possibilities. We are also currently looking into a prototype implementation of our model based on Palm devices and Bluetooth connectivity.

5 Conclusions and Future Work

In this paper, we envision the computational infrastructure of LE-communities and that there would be a proliferation of services to the user available via numerous LE-communities, and these services would continually change for a user on the move. We have given a more precise notion of the impact of user mobility on compositions of services provided by LE-communities, enabling reasoning about which services will be selected and invoked (in response to requests) as the user moves. We believe our work also forms a basis for programming the effect of users' movements on the availability of services to users via devising composition schemes. There seems to be little work in this area, but we mention ambient calculus [2] which models the notion of ambients (boundaries) that has partially inspired this work. Our work, however, differs from the ambient calculus in that we aim at modelling service compositions, and how user mobility impacts on such compositions.

There remains many avenues for interesting further work:

- We have equated LEC entry and exit to a user moving into or out of range of the network, but in practice, options should be available for the user to (virtually) exit a LEC even while within range and to remain (perhaps in a passive mode) in a LEC even if out of range.
- Not considered in the model we presented is how services of different LE-communities can be linked (independently of any requests). An interesting avenue of investigation is how the presence of several LE-communities (or their services thereof) can lead to emergent services, that is, to new services (that were not possible previously) arising when the user is in particular LE-communities (and so, has a critical mass of services). We can explore how when a user enters a new LE-community, the additional services added can be automatically linked with existing services leading to new services.
- A further extension of our work is a consideration of resource consumption for participating in a LE-community. We aim to model how a user's resources (e.g., memory on the mobile device and user attention) can be shared or distributed among the different LE-communities the user is in.

- True to the notion of community is group-based services which involve several users at a time (e.g., games or chatting applications) which we have yet to properly define in our model. We aim to study the impact of user mobility on such services, and to explore the question of LE-community rules on a user leaving or joining such services.
- There is a question of adequacy of operators. As we attempt further applications of our model, we expect new operators to be added but the operators we have presented have proven adequate for creating many applications in the logic programming context [1].
- A composition of sets of services from different LECs (as defined by a composition scheme) yields the set of services that are available to the user at a given point in time. We can complement our work with mechanims for composing services: we do not define how services are composed but rather how sets of services from different LECs are composed; once the set A of available services is defined by our composition, the user might then invoke a service composition (as defined by some service composition mechanism) or a transaction that utilizes several services in A. Each type of service might also be invoked in a composition of LECs of its own rather than having the same composition of LECs for all services.

Acknowledgements. The author thanks the reviewers for valuable comments on the paper.

References

1. A. Brogi. *Program Construction in Computational Logic*. PhD thesis, 1993.
2. L. Cardelli and A.D. Gordon. Mobile Ambients. *TCS, Special Issue on Coordination*, 240(1), July 2000.
3. Ericsson. BLIP Newsletter #7. 2001. `http://www.ericsson.com/blip/`
4. IBM. developerWorks: Web services. `http://www-106.ibm.com/developerworks/webservices/`
5. D.G. Leeper. A Long-Term View of Short-Range Wireless. *IEEE Computer*, pages 39–44, June 2001.
6. S. Loke, A. Rakotonirainy, and A. Zaslavsky. An Enterprise Viewpoint of Wireless Virtual Communities and the Associated Uses of Software Agents. In M. Rahman and R. Bignall, editors, *Internet Commerce and Software Agents: Cases, Technologies and Opportunities*, pages 265–287. Idea Group Publishing, 2001.

Client Migration in a Continuous Data Network

Anthony J. Howe and Mantis H.M. Cheng

Department of Computer Science
University of Victoria
P.O. Box 3055, STN CSC
Victoria, British Columbia
Canada, V8W 3P6
ahowe@alumni.uvic.ca, mcheng@csr.uvic.ca

Abstract. Two major issues for continuous data delivery architectures on the Internet are scalability and data continuity. Scalability is addressed by providing multiple redundant stream sources. One solution to the data continuity problem is adaptive client migration. Adaptive client migration is the ability of a client to switch from one redundant stream source to another when it experiences discontinuities in its current stream. The switching mechanism allows the service providers to adapt to changing server loads and varying jitter due to network congestion. This paper presents a client migration protocol that allows a client to migrate from one data stream to another without introducing discontinuities. We have experimentally validated our migration protocol in a simulated distributed network environment.

1 Introduction

The increase of bandwidth on the Internet, has led to a reduction of end-to-end delay in content delivery. More and more Internet applications are taking advantage of this low end-to-end delay and are delivering live content to tens of millions of users. An example of live content includes Internet radio, voice chat, and live video broadcast. Live content refers to continuous data. Continuous data requires the time sensitive delivery of data from a server to a client; late data is the same as lost data.

Two important qualities of a continuous content delivery architecture are scalability and data continuity. Scalability to support thousands or even millions of connections has been achieved through several network architectures. These architectures include server farms, proxies, and content distribution networks [1]. Each of these architectures achieves scalability by replication of the source content to multiple continuous data servers. A common problem shared by each of these architectures is their lack of guarantee for data continuity. Network congestion or overloaded servers may cause undesirable breaks or dropouts in continuous data streams. Proxies and content distribution networks attempt to improve data continuity by delivering the content closer to the end user. However, closer content is still prone to jitter caused by network congestion or server overload.

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 278–290, 2002.

This paper addresses the data continuity problem of continuous content delivery by presenting client migration. Adaptive client migration allows a client to migrate to another continuous data stream after detecting jitter or discontinuities in the current data stream. Using adaptive client migration a continuous data delivery architecture is able to adapt to changing server loads and varying jitter due to network congestion. The motivation for adaptive client migration is first presented and is followed by the client migration protocol. Finally, validation experiments demonstrate the feasibility of our client migration protocol.

2 Related Work

The *Split and Merge Protocol* of Wanjiun Liao and Victor O. K. Li [2] is different from the client migration protocol presented in this paper. The *Split and Merge Protocol* uses multiple readers and one writer to reduce the amount of bandwidth used in a continuous data network. The client migration protocol is the opposite and has multiple writers, the surrogates, and a single reader, the client. The *Split and Merge Protocol* assumes a high speed, high bandwidth local area network and does not deal with data discontinuities or jitter.

The paper *Distributed Video Streaming Over Internet* by Thinh PQ Nguyen and Avideh Zakhor describes a protocol to deliver a video stream from multiple senders to a single receiver [3]. The purpose of the protocol is to reduce packet loss and jitter by interleaving multiple streams. No two senders send the same media packet. The receiver coordinates each sender using a packet partitioning algorithm. This approach differs from our approach since we assume each sender sends the same continuous data stream.

A new architecture for delivering streaming media named Allcast has recently appeared [4]. Allcast starts with an origin streaming server and then grows as more clients connect to the network. Each client may be connected to the origin or another client receiving the stream. When a client leaves the network, Allcast must move any connected clients to other clients or the central origin. The problems Allcast addresses are very similar to ours. We cannot compare our approaches due to lack of published technical details.

3 Content Delivery

The two basic types of content delivery are discrete data delivery and continuous data delivery. Examples of discrete data include text, image, and music files. Delivery of discrete data is not time sensitive, but it is important that all the data delivered from a server will eventually arrive at a client. Continuous data is time sensitive. It must be delivered to a client at a fixed periodic rate. Late data, possibly caused by network congestion, is the same as lost data. A low data loss may be acceptable if the continuous data stream has some form of forward error correction (FEC). With FEC missing data messages can be interpolated. Reed Solomon codes [5] are one example of FEC codes.

The three basic elements of content delivery architectures include a server, client, and coordinator. The server stores content and registers all stored content with the coordinator. The client uses the coordinator to discover the location of a server that contains desired content. The server delivers the content to the client. The discovery and delivery of content is summarized by the following steps.

1. Client C requests data d from coordinator R.
2. R finds surrogates $\{S_1, S_2, S_3, \ldots, S_n\}$ that contain d.
3. R selects S_j such that S_j is the *best* surrogate to supply d to C.
4. R connects C to S_j.

Steps 1-3 handle data discovery and step 4 handles data delivery. The measure of *best* when determining a server for a client may be based on one or more of the following properties: the geographic location of a server with respect to a client, the network locale of a server and a client, the number of hops from the client to a server, the current network resources of a server, the current computing resources of a server, or the quality of the *data* channel between a server and the client. In choosing the *best* server for a client the primary objective is to optimize the quality of service (QoS) to the client. An optimized QoS for discrete data means that the discrete data will arrive sooner at the client. An optimized QoS for continuous data means reduced jitter and reduced data discontinuities for the client.

Commercial discrete data delivery architectures that use the above discovery and delivery steps, with a minor change in step three, include FTP, Napster, and Morpheus. The minor change in step 3 is that client C picks the best server S_j for itself from a list of servers provided by the coordinator. The coordinator for FTP may be a web based search engine, whereas the coordinator for Napster and Morpheus are nodes dedicated to the sole purpose of data discovery. In each of these architectures a client may be a server and a server may be a client.

To improve QoS, each of the three discrete data delivery architectures mentioned above provides methods for receiving a discrete data item from multiple servers. With the ability to receive from multiple servers a client can receive half of a data item from one server and then switch to another server and receive the last half of the data item. A client would switch from one server to another to increase the speed of the data item delivery. Mechanisms inside Morpheus are more advanced and allow pieces of a data item to be simultaneously delivered from multiple servers, thereby speeding up delivery.

The delivery of data from multiple servers is considerably more interesting for live continuous data. An improved QoS cannot be achieved by downloading future data from multiple servers, as was done in Morpheus, since the future live data does not yet exist. However, the ability of a client to switch between servers could improve the QoS. When the QoS deteriorates from one server the client could migrate to another server to obtain a better QoS. In addition the coordinator could use client migration to balance client load among servers thereby increasing the chances of a higher QoS in the overall continuous data delivery architecture.

Due to varying distances from the client to multiple continuous data servers, the servers will be at different latencies from each other. In migrating from one stream to another, client migration must handle these varying latencies without causing jitter or introducing discontinuities in the data stream. These issues are the topics of the next section.

4 The Client Migration Protocol

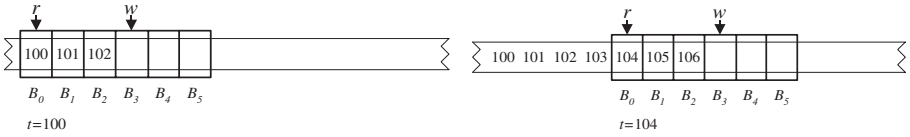
The delivery of continuous data requires the periodic timely delivery of data messages from a producer to a consumer. The consumer consumes these data messages at the same periodic rate as they are produced. Late data is the same as lost data. The timely delivery of continuous data messages is difficult on the Internet due to unpredictable congestion. Congestion introduces jitter and delay, that as a result, causes lost data. One method for dejittering a data stream is to use a buffer. On initial connection to the producer, a buffer is partially filled with data messages prior to their being consumed. Therefore when a data message arrives late there are some data messages in the buffer available for consumption. However, consecutive late data messages will cause the buffer to empty and lead to a buffer underrun. If a buffer is too small, a burst of data messages may cause the buffer to overflow.

4.1 A Buffer Model

We model a buffer with a sliding window. The window encapsulates a range of M data message cells labelled B_0 to B_{M-1}. The window has a read head labelled r and a write head labelled w. The values of the read head and the write head specify their relative location within the window. Assuming the data stream is ordered and reliable, a time counter t is maintained to specify the *location* of the beginning of the window within the data stream. Figure 1 shows the window at time $t = 100$ and at time $t = 104$. In general, at time $t = k$, $B_0 = k, ..., B_i = (k + i)$ for all $i < w$, and $B_{k+j} = empty$ for all j where $w \leq j < M$.

The counter t is incremented after every *read* operation and the window is slid to the right by 1. When the window slides to the right and $w > 0$, w is implicitly decremented by 1 to remain at the same absolute position within the data stream. This change in t and w can be seen after a data message is read in Figure 2. When w is between 0 and $(M - 1)$ an incoming data message is written to B_w and w is incremented by 1. This change in w can be seen after a data message is written in Figure 2. A window is underrun if $r = w = 0$ when reading. A window is overrun if $w = M$ when writing. Underruns and overruns of the window cause discontinuities in the data stream.

Figure 3 shows the use of sliding windows for each network element of a continuous content delivery network. An origin delivers data to two surrogates and then each surrogate delivers data to a client. The origins and surrogates have a window size of two and the clients have a window size of six. The data



(a) At time $t = 100$, $B_0 = 100$, $B_1 = 101$, $B_2 = 102$, $B_3 = B_4 = B_5 = empty$

(b) At time $t = 104$, $B_0 = 104$, $B_1 = 105$, $B_2 = 106$, $B_3 = B_4 = B_5 = empty$

Fig. 1. The sliding window at time $t = 100$ and at time $t = 104$

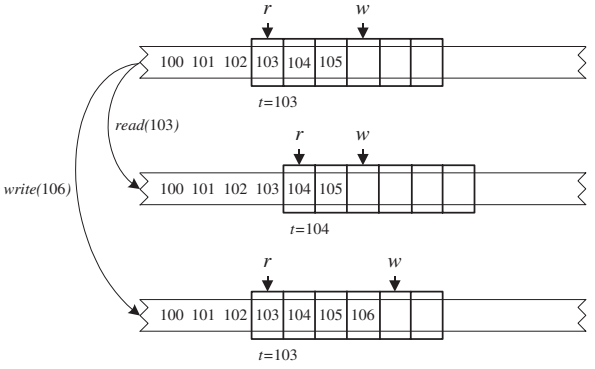


Fig. 2. The actions of a read operation and a write operation on a sliding window

messages that are in transit between the elements in Figure 3 illustrate network latency. Surrogates may have different network latencies in their data paths to the origin. This latency difference can be seen by the different values of time t for the surrogates S_1 and S_2. Surrogate S_1 has a higher value of time t than surrogate S_2 meaning that S_1 has less latency to the origin O. Latency differences are further propagated to the clients. Client C_1's value of time t is greater than client C_2's value of t. The role of the coordinator is for initiation of client migration. Similar terminology of coordinator, origin, surrogate, and client is defined in [1].

4.2 Client Migration

The migration from the existing surrogate stream to a new surrogate stream requires the splicing of the two streams to form a continuous stream. As discussed previously surrogates may have different time t values due to network latencies. The stream splicing algorithm must take latency differences into account and

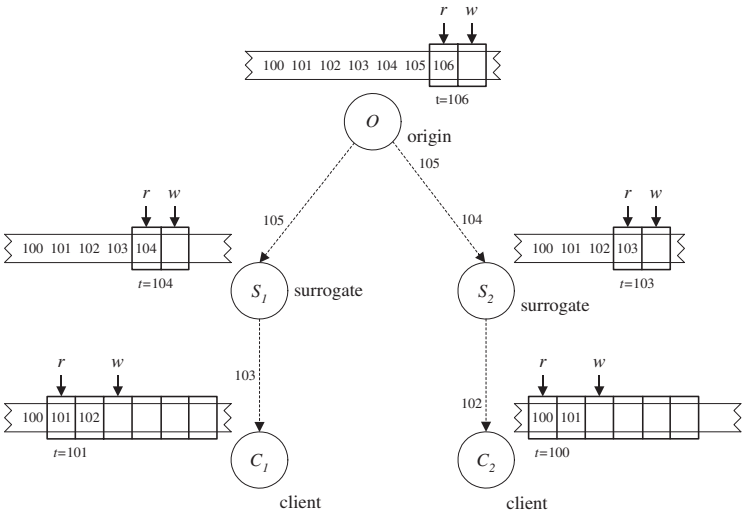


Fig. 3. Data channel communication in a continuous data network

splice the two streams to preserve the monotonically increasing order of the sequence numbers without introducing discontinuity. In order for a migration to succeed the following three steps must take place:

1. an acceptance test - *whether* a new data stream is acceptable;
2. the splice point check - *where* to splice the new data stream; and
3. a continuity test - *when* the old data stream is finally spliced with the new data stream.

The acceptance test determines if a new data stream fits within the bounds of the client's current window. A client is limited to the surrogates that it is able to migrate to by the following properties: the client's window size M, the client's value of time t, and the new surrogate's value of time t.

Figure 4 shows a client and five surrogates that have different values of t. The client C is currently receiving data from surrogate S_1 and has a t value of 103. Examination of the figure shows that C may only migrate to surrogates S_2 and S_3. The time t values of surrogate S_2 and S_3 fit within the bounds of C's window. Migration to surrogates S_4 and S_5 is impossible since the t values of each of these surrogates are out of the bounds of the client's window. In general the migration of client with time $t = k$ to a new surrogate with time $t = x$ is only acceptable if $(x - k)$ is between 0 and $M - 1$. The t value of a surrogate is determined by the sequence number of the first data message from the new surrogate's stream. If the stream is accepted, the splice point becomes $s = (x - k)$.

After a successful acceptance test, a decision must be made on where to begin saving the new data stream in the window. This is referred to as the splice

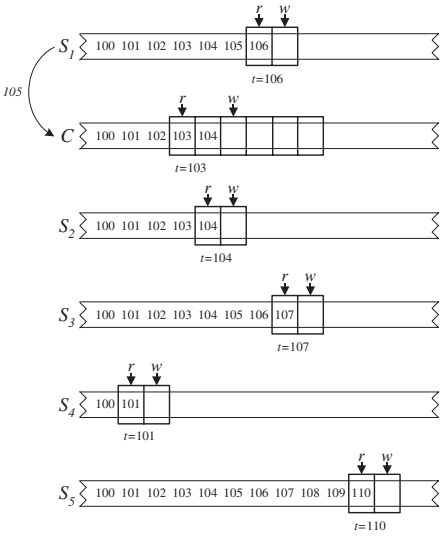


Fig. 4. Surrogates and a client at various positions in a data stream

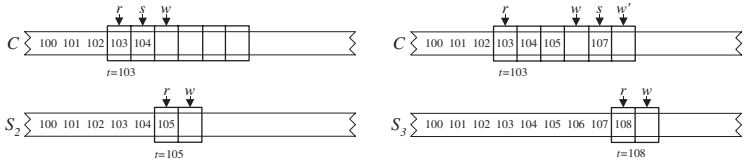
point check. The value of the splice point specifies its relative position within the window. Using the splice point $s = (x - k)$, discussed in the previous paragraph, there are two cases for the relation between s and the client's write head w:

1. $s \leq w$ - the splice point s is *at* or *earlier* than the write head w; and
2. $s > w$ - the splice point s is *later* than the write head w.

Both of these cases are illustrated in Figure 5 by showing the placement of the splice points for client C's migration to S_2 and S_3 of Figure 4. Both S_2's and S_3's windows have advanced within the data stream since each surrogate has sent an initial data message. The initial data message is written to cell B_s.

Once the splice point is chosen a final test for data stream continuity must be performed. A client's data stream becomes continuous when $w \geq s$; this is known as a successful splice. For the first splice point case where $s \leq w$ the data stream is automatically continuous, the messages in the window from $B_{(s+1)}$ to B_w are discarded, and the write head w is reset to $s + 1$. For the second splice point case where $s > w$ there is a gap between B_w and B_s meaning that the data stream in the window is not continuous. A write head for the new stream labelled w' is situated at B_{s+1}. The stream is not continuous until enough data messages arrive from the current surrogate so that $w = s$. When the stream finally becomes continuous the write head w is reset to w'.

The client migration protocol consists of messaging and stream splicing. The messaging required to perform the migration occurs on the *control* channels of the following network elements: the client, the coordinator, the client's existing



(a) A splice point s is *earlier* than the write head w

(b) A splice point s is *later* than the write head w

Fig. 5. Client C's migration to the valid surrogates S_2 and S_3

surrogate, and the new surrogate to be hosting a data stream to the client. Stream splicing, as discussed previously, is concerned with the splicing of the existing surrogate's data stream and the new surrogate's data stream that are both delivered on the *data* channel. The *control* channel and the *data* channel are orthogonal; i.e. the messages of either channel are independent from each other.

Figure 6 shows a message sequence chart for the client migration protocol. Both control and data channel communication are shown. In this figure the coordinator R initiates the migration of the client C from its existing surrogate S_1 to the new surrogate S_2. The new surrogate S_2 is the best alternative surrogate for C. Seven messages are required for a complete migration when initiated by the coordinator. When a surrogate or client initiates migration an eighth request message is required to be sent to the coordinator. In this figure S_2 had an initial sequence number *later* than S_1 and required d_6 to make the buffer continuous.

4.3 Fault Analysis

Client migration is necessary to transfer a client from a surrogate of poor QoS to a surrogate with better QoS. An existing surrogate with poor QoS may cause control or data message faults during migration. Control or data message faults may cause discontinuities in the increasing monotonic order of the messages within the window. Up to this point it has been assumed that the data stream was continuous and had the property such that the datum to the right of d_x was always d_{x+1}, where the subscript represented the sequence number. For this section the requirement is that the sequence numbers in the stream be continuous and in strictly increasing order.

The continuous requirement of the buffer means that the data stream inside the window must contain no gaps. Gaps may occur when the existing stream faults during the splicing of a later sequence numbered data stream to the existing stream. A gap can be removed by discarding the cells that make up the gap. Figure 7 shows the removal of the gap after an existing stream fault. After

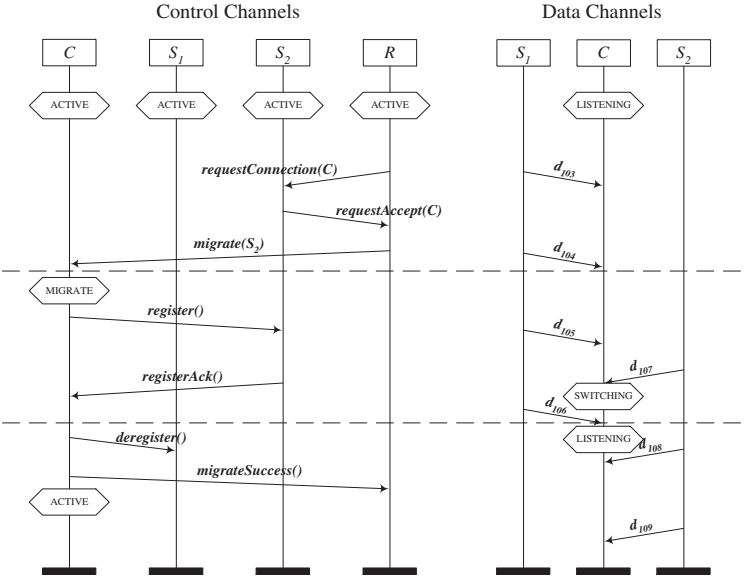


Fig. 6. The message sequence chart for a client migration

the removal of the gap, the write head w is reset to w' and the new data stream continues to fill the buffer.

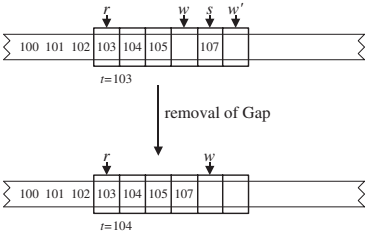


Fig. 7. The removal of a gap after a failed migration to a *later* stream

Strictly increasing means that the data message to the left of d_x is always d_{x+y} where $y \geq 1$. Figure 8 shows a window with sequenced messages in strictly increasing order. Notice how message 5 has been lost and that message 6 has been written in the cell where message 5 should have been. It is up to the consumer of the window's data messages to handle the discontinuities in sequence numbers

through the use of error correction or other means. The consumer will adjust its *get* frequency on the buffer according to missing sequenced messages.

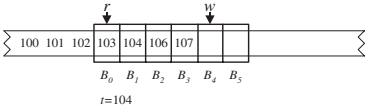


Fig. 8. Non-monotonically increasing sequence numbers of the data stream

The strictly increasing property means that B_r will not be equal to time value t. Providing that $w > 0$ the value t is found by using the sequence number of the datum in B_{w-1}. The value of t is then $B_{w-1} - (w-1) = B_{w-1} - w + 1$. If $w = 0$ then the value of t is the last consumed sequence number incremented by 1. The t value in Figure 8 is not the same as the sequence number in cell B_r. This value of t always ensures that the splice point always inserts the new stream such that the strictly increasing order is preserved. For example if the first data item of the new stream has a sequence number 6, for the window in Figure 8, using a t value of 3 from $t = B_r$ would place it at cell B_3. This would cause violation of the strictly increasing property of the stream since 6 already exists at B_2. However if t is calculated in reference to B_{w-1} the datum with sequence number 6 would be placed in cell B_2. A splice point will always preserve the strictly increasing order since the t value calculated in reference to B_{w-1} assumes a monotonically decreasing stream going left from B_w.

Control message faults include lost control messages, duplicate surrogate migration requests, or logout from either the existing or new surrogates during migration. Timers on each of the network elements involved in migration provide detection for lost control messages.

Data message faults include messages that are late or lost. Consecutive lost messages may be due to the failure of the stream or failure of the surrogate serving the stream. The solution to solving late messages was to use a buffer. Lost messages have not been a concern up to this point since it has been assumed that the data channel has been ordered and reliable. For this section the reliability property of the data channel is removed, leaving an unreliable data channel that delivers messages in order. If every incoming data message is written to the next empty cell, provided that space is available, then the window will keep the required continuous and strictly increasing sequence number order.

Using the method discussed previously for calculating t on a non-monotonic sequenced stream the acceptance test and the splice point check steps of client migration remain unchanged. If $s \leq w$ then the data continuity test will pass and migration is complete. However, the data continuity test step must be changed when $s > w$. When $s > w$ three conditions can be used to test for whether a data stream is continuous in the presence of lost messages:

1. test for $w = s$;
2. test for $B_w \geq B_s$; and
3. reset the client's timer for $T \times (s - r)$.

The first test for continuity becomes true when there are no lost messages in the existing stream during migration. The second test becomes true when there are lost messages on the existing stream during migration. Similar to the situation shown in Figure 7 the gap is removed and the write head w is reset to the new stream's write head w'. The third test is the maximum time the client can wait for the existing surrogate to overlap with the new stream. The variable T is the period of data delivery.

The quality of the new stream is difficult to determine. The coordinator's measure of *best* may not lead to a data stream with a better QoS for the client. The time taken for the migration to complete may not be enough to determine the QoS of the new stream. Migration may need to be requested over and over before a data stream with an acceptable QoS is found.

5 Experimental Validation

Experiments have been created to demonstrate that the client migration protocol has the following three properties: distributed initiation, idempotency, and resiliency. The migration protocol experiments were implemented in the coordination language COOL [6]. COOL is an object based language that facilitates the creation of distributed applications. The experiments were executed on Pentium level computers running a combination of Windows and Linux. The computers were interconnected in a 10Mbps ethernet network; COOL used UDP/IP for communication between computers. Experiments were distributed across multiple machines.

Distributed initiation is the ability of any entity, except the origin, to initiate migration. A coordinator may initiate a client to migrate after monitoring network load. A server may initiate migration if it is overloaded. A client could detect a deterioration of the quality of its data stream and initiate migration. Due to the nature of the client migration protocol, multiple requests for migration may occur simultaneously.

Once it was demonstrated that distributed initiation was possible, an experiment to show an automatic migration was conducted. An automatic migration would occur when the client experienced jitter on its current stream. To demonstrate the automatic migration a jitter detection mechanism was added to the client. The experiment first started by the client receiving continuous data from one surrogate. A load was initiated at the surrogate to introduce jitter in the data stream. Immediately the client detected the jitter, requested migration, and was successfully migrated to another surrogate. Automatic migration showed that the client migration protocol could be adaptive.

An important property when using an asynchronous message based network is idempotency. An operation is idempotent if multiple initiations of the same request result in a single request. It is important that the migration of a client

from one surrogate to another is idempotent. For example, if a client is currently migrating, any additional requests to migrate are ignored. Idempotency must occur at the coordinator and at the client. Idempotency at the coordinator is necessary to handle multiple migration requests from a surrogate and a client during the migration of a client. Idempotency at the client is necessary to handle multiple migration requests from the coordinator during and after a client's migration. Multiple requests may come from the coordinator if the coordinator does not receive a response from the client that a migration had occurred.

The migration protocol is resilient and can handle varying surrogate latencies, lost control messages, and some data stream faults. The migration protocol can splice a new data stream that is upstream (*later*) or downstream (*earlier*) from the current data stream. If the new stream is too far upstream or downstream to fit within the bounds of the client's buffer the migration protocol will abort the migration and report to the coordinator. Furthermore, lost control messages will not cause any network entity to deadlock or lose resources.

The time it takes for one client migration is dependent upon the fixed cost of the control messages and the variable cost of filling the gap between the existing data stream and the new data stream. A maximum of eight control channel messages are required for the initiation of a migration. If there is a gap after the admission test the cost for the migration to finish is $n * c$ where $n = (s - w)$ is the size of the gap and c is the computation time of examining the sequence number of every message from the existing stream to ensure it is not greater than or equal to the sequence number of the message at the splice point.

6 Conclusion

For many years discrete data delivery architectures have been able to improve reliability of the retrieval of a data item by allowing the data item to be received from multiple sources. A logical step for this feature is to apply it to continuous data delivery architectures to reduce data discontinuity and improve QoS. The client migration protocol and experimental validation described in this paper demonstrated the feasibility of a client to receive continuous data from multiple redundant sources while maintaining an acceptable QoS. Furthermore the adaptive property of the migration protocol adds a new level of resiliency to clients that previously did not exist with current continuous data networks. Future work for this research includes the integration of the client migration protocol into a real continuous data application that can be used on the Internet such as Internet audio.

Acknowledgements. Support for this research came from the Natural Sciences and Engineering Research Council (NSERC) of Canada, the Mathematics of Information Technology and Complex Systems (MITACS) of Canada, and Nortel Networks.

References

1. G. Tomlinson M. Day, B. Cain and P. Rzewski. A model for content internetworking. Technical report, Network Working Group Internet-Draft, November 2001.
2. Wanjiun Liao and Victor O. K. Li. The split and merge protocol for interactive video-on-demand. *IEEE MultiMedia*, 4(4):51–62, October–December 1997.
3. Thinh PQ Nguyen and Avideh Zakhor. Distributed video streaming over Internet. *Proceedings of SPIE Conference on Multimedia Computing and Networking*. January 2002.
4. Andy Oram. Allcast: New life for live content. *O'Reilly & Associates Inc*, July 2001.
   `http://www.openp2p.com/pub/a/p2p/2001/07/17/allcast.html`
5. Stephen B. Wicker. *Error control systems for digital communication and storage*. Prentice-Hall International, Inc., 1995.
6. Mantis H.M. Cheng, Gordon W. O'Connell, and Paul Wierenga. COOL : A concurrent object coordination language. Technical Report DCS-267-IR, Department of Computer Science, University of Victoria, Victoria, BC, Canada, July 2001.
7. Anthony Howe. Client migration in a continuous data network. Thesis, Department of Computer Science, University of Victoria, Victoria, BC, Canada, February 2002.

Using Jini to Integrate Home Automation in a Distributed Software-System

Peter Rigole, Tom Holvoet, and Yolande Berbers

KULeuven, Department of Computer Science
Celestijnenlaan 200A, B-3001 Leuven
Belgium
{peter.rigole, tom.holvoet, yolande.berbers}@cs.kuleuven.ac.be

Abstract. The last few years, a tendency arose to integrate various programmable devices through ever expanding computer networks. One particular domain in which this evolution stood out clearly is home automation. The EIB[1] standard defines a home automation solution that consists of a network of cooperating components with little computational power. Bridging the EIB network with a computer network allows software to interact with these EIB components. However, past attempts to link EIB components with computer networks fell short in dynamism, automatism or user friendliness. In this paper we present a distributed software framework that overcomes these issues, and is capable of automatically generating a software model based on the configuration of the components on the EIB fieldbus. A framework that can be used in this perspective is Jini. Its excellent capabilities for dynamic reconfiguration and its proven deployment in domestic and office environments make it an appropriate candidate for supporting home automation systems.

1 Introduction

Home automation is well known as a technique to create a more comfortable living environment for inhabitants of modern buildings and to fine-tune these conveniences according to their desires. Classic systems usually work just the other way around: the inhabitants must adapt themselves to their surrounding environment because the electric installation was integrated statically when the building was constructed. Home automation typically controls illumination, shutters and heating, but also security, power saving and economic use of energy.

Till now, domotic[2] installations are typically closed systems in which the components only interact with each other using some low-level communication protocol. Few software environments exist that can communicate with the home automation components, but none that meets the needs of nowaday's network and ubiquitous computing availability. We developed a framework that brings a new approach to the integration of small embedded devices like domotic components

[1] European Installation Bus
[2] Domotics or home automation. We use these terms interchangeably in this paper

J. Plaice et al. (Eds.): DCW 2002, LNCS 2468, pp. 291–303, 2002.

into a distributed computing environment. The EIB home automation system is introduced in the following subsection, followed by our system-requirements and a description of the interconnection between home automation and software.

1.1 Home Automation

Home automation usually consists of various kinds of small devices which are interconnected via a fieldbus. The EIB domotic system we use, is a decentralized system where each component directly addresses its destination components via a communication protocol that uses multicast messages. These messages allow a component to control multiple other components with one single telegram[3]. Special group addresses are used to address a collection of domotic devices.

Each domotic component consists of two parts: its bus coupling unit and the end-user hardware component, which is usually referred to as the application module (AM). This can be a switch, a relay, a thermometer,... The bus coupling unit takes care of the communication on the EIB-bus and is used by the end-user hardware to interact with the other domotic devices. To perform this interaction, the BCU has access to addressing tables, stored in its built-in flash[4] memory. These address tables are defined in the first part of the memory and are bound to the solid specifications of [1]. The remaining part of the flash memory can freely be used by the application program[5] designers to store user-configurable parameters used to fine-tune the application module. Figure 1(a) shows an example of an application module (a two-button switch) and its corresponding bus coupling unit. The two parts are sometimes build in one piece.

The group addresses in an address table are grouped in a number of sets and each set is assigned to one communication object, which is a concept that reflects the various functionalities of the application module in its BCU. objects is able to send a telegram to a group address and/or to receive telegrams from one or more group addresses. A simple example might clarify how this works. Figure 1(b) shows a possible configuration with two lights and a switch. A double relay (a binary output component) with built-in BCU switches the two lights. The BCU's application program has two communication objects, one for each relay. The first communication object (C1) listens to the group addresses 1/2/0 and 1/2/2, the second (C2) listens to 1/2/1 and 1/2/2. When the switch now sends a telegram to group address 1/2/0 with data value 1, then the relay will switch the first light on. If the switch sends a telegram to 1/2/2 with value 0, then both lights will get switched off, because their communication objects both listen to 1/2/2.

The part of the application program that can be implemented freely by the manufacturers of domotic components typically defines extra parameters that are used to configure the application module. A relay with a timer for example, needs to be configured with the time period to leave the light on. Being the main

[3] A telegram is a message entity on the EIB fieldbus, like a packet in TCP/IP networks
[4] Flash or EEPROM, electrically erasable programmable read only memory
[5] The application program is the name for the whole program in the flash memory

(a) A BCU and AM (b) A button that switches lights.

Fig. 1. Domotic components

access point to the component, both the addressing tables and the free part of the BCU is readable and reprogrammable via the fieldbus itself, which allows us to add automatic configuration extraction and reprogrammability features in our framework. This way, the framework is able to generate a software representation of the domotic network, based on that memory content. Unfortunately, since there is no standardisation for the memory location of additional parameters, the content of the free part is useless without external information about it. Its management is in consequence not generalizable, which limits the functionality of a generic software model to the level of configuring communication objects and their group addresses. We go beyond the generic part by adding object request broker functionality that loads extensions of the basic framework based on the signature of the application program in the flash memory.

The content of the flash memory of components is accessible through the EIB bus by using a software driver[6] and a serial interface connecting a PC to the EIB bus. EIB devices are then addressed by using a unique physical address.

1.2 Software Integration

The framework we present in this paper is able to fully integrate the functionalities of a home automation system into a distributed software environment. We achieved this by proposing the following essential requirements before we began:

- The framework should offer all the functionalities of the components on an EIB home automation system to software or to users on a computer network (switching lights, moving shutters, controling the heating,...)
- The framework should allow one to reconfigure and reprogram the devices of an EIB domotic system (allocating new group addresses to a communication object, adjusting parameters,...)

[6] We used the driver of JNet Systems[2]

- The framework must be independent of a particular domotic configuration (e.g. the configuration in some building).
- The framework must be dynamic. When one changes the home automation system (e.g. by adding a new component), the framework should take these changes into account and adjust the software representation to reflect them.
- The framework must work automatically, which means the software representation of a concrete EIB configuration is not defined by hand, but automatically generated by the framework instead.
- The framework should work in a distributed environment.
- The framework should offer user friendly interfaces to its users.

Three important solutions have already been worked out to construct a bridge between EIB home automation and a software environment. The EIB Association (EIBA[1]) was the first to develop a tool, the ETS tool, which is mainly used to program and configure EIB compatible devices. It offers services only to the computer on which the tool runs. Its successor, the brand new version of ETS, called iETS, also offers Internet connectivity towards the EIB bus. However, the main goal of iETS is to allow remote maintenance on EIB networks via TCP/IP and not to make the functionality of the underlying domotic components available to software connected to the computernetwork. The ETS tool needs a project file of an EIB configuration to adapt the application programs of its components, which makes it unfit for use in an unknown EIB environment.

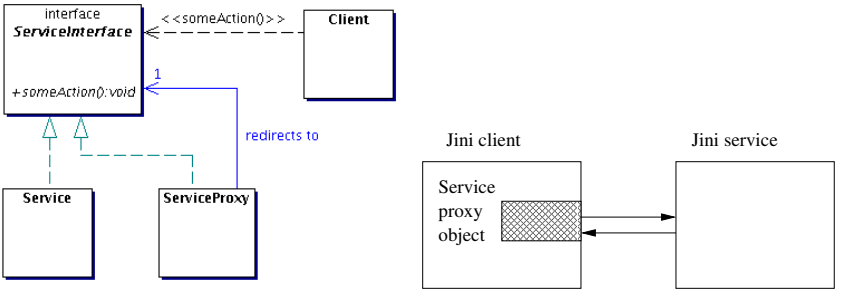
The second framework was built by a company called JNet Systems[2]. They developed the EIBLET model, which is a high-level Java API[7] for building agents that control one or a collection of domotic devices in a building. Since a programmer must construct new agents for each new home automation installation, we can't call the EIBLET framework neither dynamic nor automatic and definitely not independent of a concrete domotic configuration.

The last framework is a Java implementation originating in the work of Wolfgang Kastner and Christopher Krügel[3] about Jini connectivity for EIB networks. They used Jini[8] to offer services in a distributed manner. They also worked out an agent-based solution in which each agent is a service on the network that handles a set of home automation functionalities. An agent might for example manipulate the illumination in a room (*lightAgent*). As you can see, there is a similarity with the EIBLET solution and it meets our distributed requirement, although there is no API provided to construct agents. Each new agent needs a fresh implementation, even the Jini part needs to be reimplemented each time. In other words, the agents have to be build on demand of the inhabitants of the building. Again, this solution lacks some degree of dynamism and automatism.

Unfortunately, none of the existing solutions fully meet our suggested requirements. In the following section, section 2, we start with a description about how Jini works and how it can be used to implement distributed software. We explain

[7] Application Programming Interface
[8] More details about Jini will be discussed later in this paper.

how we modelled our framework in section 3, followed by some Jini related work in which we compare Jini to the Web Operating System[4]. The Web Operating System also presents a way to build distributed software like Jini, but there are some conceptual differences which lead to a complete different implementation. Finally, a conclusion is given in section 5.



(a) The proxy design pattern (b) Controlling the service

Fig. 2. Using the remote proxy design pattern

2 Jini

2.1 About Jini

Jini[5] is a framework written in Java and launched by Sun Microsystems almost three years ago. It is designed to build distributed network applications in an easy way. The services in a Jini community[9] announce themselves on the network dynamically: they come and go as they like.

A central service on each network, the lookup service, keeps an actual view of its surrounding services by requiring that each Jini service seeks a lookup service to register itself with. The lookup service is found by means of a multicast discovery protocol on the local network. A leasing mechanism ensures that there are no invalid references to services. When a lease expires before it gets renewed, the service is considered to be unavailable and is removed from the lookup service. The global result is a federation of spontaneously joining and leaving services.

The registering procedure for services and the lookup procedure for clients is done via method calls on a remote Java object, which is obtained as the result of a successful discovery. When a service registers with a lookup service, it gives the lookup a serialized[10] proxy object that has a remote reference to the service. The

[9] A Jini community is usually called a federation

[10] Serializing means transforming an object into a byte stream

remote proxy design pattern [6] used here is illustrated in figure 2(a). The clue is that the service is always referred to as if it were its interface, so the proxy can play the role of a local representation for the service. The proxy simply redirects every method call to its service via remote method invocation. Figure 2(b) shows us the resulting situation: the client has direct access to the services it needs.

A clever query mechanism allows clients to find the services they need via the lookup service. Three kinds of elements can be combined in the query to seek a service. The unique identifying number of a particular service is the first option. This is the easiest way, given that you know the number of the service you want. The second way is to search based on the interfaces of the services you are willing to find. The interfaces implemented by Jini services, are very important because they define the type of the service and the way to interact with them from within clients. The final alternative is a template-matching query mechanism in which you use attributes to search with. Attributes are objects attached to Jini services which describe the service. To perform the query, you just have to define the query boundaries by instantiating a template-attribute with the values you want the service to have and use it in the query. The last two options are powerful query mechanisms, because the polymorph nature of interfaces and attributes can be utilized to narrow or broaden the search. If you have an inheritance structure where *ColorPrinter* and *BlackWhitePrinter* inherit from *Printer*, then both color printers and black and white printers are found when the interface for a regular printer is used. Queries like *'find all color printers that do 600dpi, located at the second floor of our department'* are a walk in the park for Jini when the services are modelled well.

2.2 Using Jini

It should be easy to meet the distributed requirement in a system by putting the Jini-engine under the hood of the framework. However, in reality, this doesn't seem to be true. The reason is there is no high-level API available for Jini. Hence, lots of code has to be written all over again each time one wants to create a Jini service or client. Because this would lead to a large overhead in code and development time, we wrote a small framework that can easily be reused for each Jini service. The framework contains one basic class (*BasicService*), which has to be extended by each new Jini service. This extending class should implement the *'protected Object createProxy()'* method, which returns the proxy object when lookup services ask for it. This returned proxy object should, of course, apply to the pattern shown in figure 2(a). The interface on top of that figure has yet to be defined to complete the pattern. It defines the way to interact with the service and should be provided by the designer of the new service.

All other functionalities (discovery, lookup, leasing,...) are taken care of by the *BasicService* and an extendible default implementation is present for the important Jini related features (e.g. default attributes). The framework also implements the standard administrative interfaces of Jini, so that administrative tasks can be performed remotely on the service. A recovery mechanism is built-in as well. It saves the volatile data of each service automatically to disk every now

and then. This data can be used at startup to initialize the service. The result is a small framework that allows us to write a Jini service with very little code.

We created an analogue framework for Jini clients in which only the kind of service you're interested in has yet to be defined. When that service is then found via a lookup service, its proxy is past as a parameter to a method that should be overloaded within the new client.

3 Modelling Home Automation

Our framework is based on an extendible model, which describes a domotic component from its lowlevel basics: its application program. The addressing tables and the communication objects of the application program are the common denominator for all EIB home automation devices, while their application module can be arbitrarily complex. More details about the design aspects are given in de following subsections.

3.1 Layered Structure

The framework has a layered structure that reflects the different levels of abstraction in the model. Figure 3 nicely displays this structure. Services from one level use services of other levels. This way, all concepts remain logically separated. We continue with a description of each level.

Bus Communicator Service. This service is offered on a machine with a physical connection to an EIB fieldbus and takes care of reading and sending telegrams on the bus. By adding a remote listener to this service, interested entities can get notified when a new telegram arrives. It is also possible to read the memory content out of a component's flash memory, or to write new data into it. We used the buscomunicator API of JNet Systems [2] to implement this bridge.

Busdevice Service. This service is actually the core of the framework. Each component on the EIB bus will eventually be modelled by this busdevice service, or by one of its more concrete subclasses. It offers all the functionalities of the real component. A binary output for example will be able to send an 'on' or 'off' message to each of its outputs. The possibility to reconfigure the EIB device (e.g. changing group addresses connected to communication objects, adjusting parameters) is also provided at the level of this service. Because the default implementation of this service can't contain the component specific configuration details (like its parameters), subclasses that know more about the content of the manufacturer's application program will often be used. The busdevice service utilizes the underlying bus communicator service to interact with the fieldbus. Section 3.2 reveals more implementation specific details about this service.

Service Controller. Since the framework needs to start a matching busdevice service for each component on the EIB bus, there is a need for some controller to assign this task to. A service controller is used to keep the consistency between the state of the devices on the fieldbus and the software services. It is able to scan all physical addresses to check which components are available. When a component has not yet a service running somewhere on the network, the service controller can then initialize a new service for it, based on its memory content. Because of performance reasons, the service checks first whether there is a recent version of that memory available on persistent storage. All this ensures that the system can start up services **automatically**, whitout human intervenience.

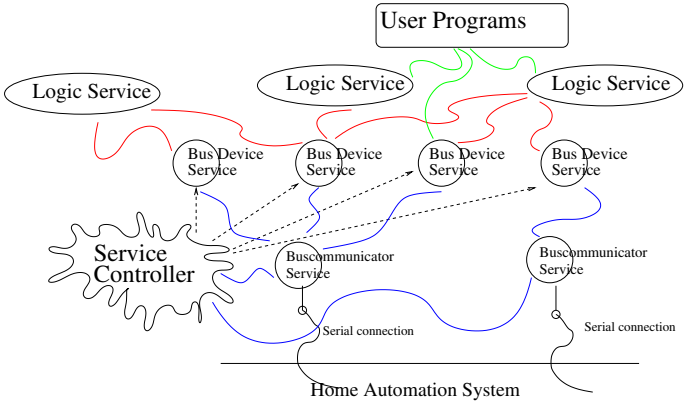


Fig. 3. The hierarchy of EIB bus services

In addition to this automatic-property, occurs in a **dynamic** way. A library with application programs and a classloader are put in action to start the corresponding concrete implementation. When no such implementation is available, the default busdevice service is used, which offers the functionalities of an EIB device at the level of its communication objects. The component's memory content is read out and interpreted to initialize this Jini service.

Logic Services and User Programs. The layer of logic services presents a real world representation of some services to create an easier to understand view for the users of the framework. A binary output for example is actually a relay, which can switch something on or off, but the device it switches is not known at the lower levels. A logic service covers this issue. It models the concrete physical situation behind some of the components. A logic service might, for example, control all the shutters in a room or building, or the lights in the living room. The top-level layer on figure 3 consists of user programs, they are end-user applications that use all or parts of the framework.

3.2 Implementation Aspects

The *BusDevice* class has a central position in our busdevice model. provides a *BusDeviceProxy* and a default interface, which is already enough to make our component Jini-enabled. By employing a *BusCommunicatorClientFactory*, the busdevice service can generate a client that communicates with a buscommunicator (cfr. the bottom of figure 3). As facade[11] for an EIB component service, it brings all the functionalities of the component (sending data to certain group addresses, adding listeners to communication objects, programming the component,...) to the open world of Jini federations via a general interface, which is available in the framework for busdevice clients. Each busdevice also automatically adds some GUI-attributes to its federation. Clients can instantiate them to obtain a nice graphical access point to the administrative tasks on the service and to the communication objects of the device. All communication objects are listed in the graphical representation, together with their data value. The data of a communication object that is allowed to send[12], can be changed and sent on the fieldbus with a simple click on a button. Figure 4 shows a client that instantiated the GUI-attributes of two services.

The *BusDevice* conceals an *ApplicationProgramController* object that handles all application program related features. It has a reference to an *ApplicationProgramInterpreter* object which is able to convert the raw binary representation of an application program to *CommunicationObject* objects. These are then linked to the group addresses (*GroupAddress* objects) they listen to, or to which they can send telegrams. Both group addresses and communication objects are also used as attributes of the Jini service. This way, by using them in their lookup procedure, clients can easily look for services that send or receive telegrams to certain group addresses.

Beside a reference to an interpreter, the controller also has a reference to an *ApplicationProgram* object. This object is useful to change the application program in its high-level representation, what makes it easy to let communication objects send or listen to new group addresses, to adjust parameters,... Once changed, the object can easily be transformed to the raw binary format by the interpreter again, ready to load into the BCU's flash memory.

As you might think now, this framework does not cover the functionality of all domotic components, but only what they all have in common: their protocol to send telegrams to each other. Although this covers the better part of the interaction, there is still a need to offer interfaces that are more adjusted to the underlying application program. Therefore, concrete bus device services should extend the *BusDevice* class and let it offer easier to use interfaces that allow the user to change the component's parameters, or it can provide easier to understand methods (e.g. '*switchRelayOne()*' when the component is a binary output). To meet this need, and because Jini requires standarized interfaces, we created a hierarchy of interfaces which classifies the components as it is done in the libraries for the ETS Tool. also not uncommon to extend the graphical

[11] cfr. the Facade design pattern ([6])

[12] e.g. a communication object of a sensor like a button or a thermometer

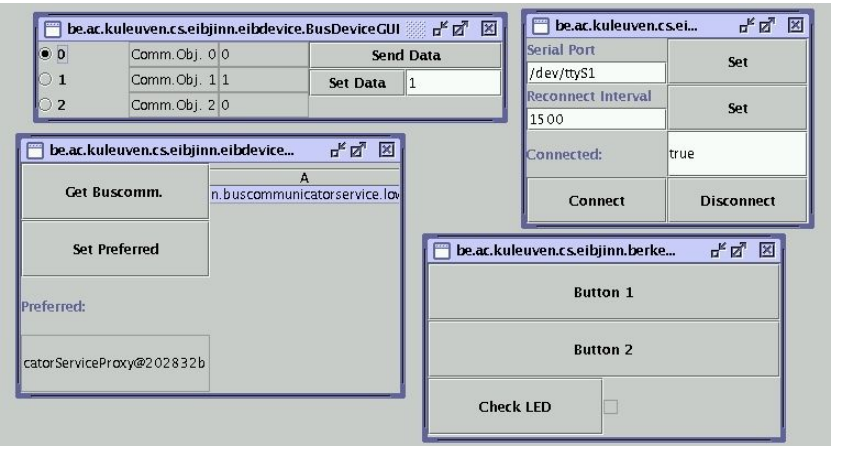


Fig. 4. GUI attributes

attributes to allow easier user interaction with a component. Mind that these subclasses of *BusDevice* should be added to the library that is used by the service controller (cfr. figure 3), to ensure these classes can be loaded and initialized by the controller.

3.3 Simulation

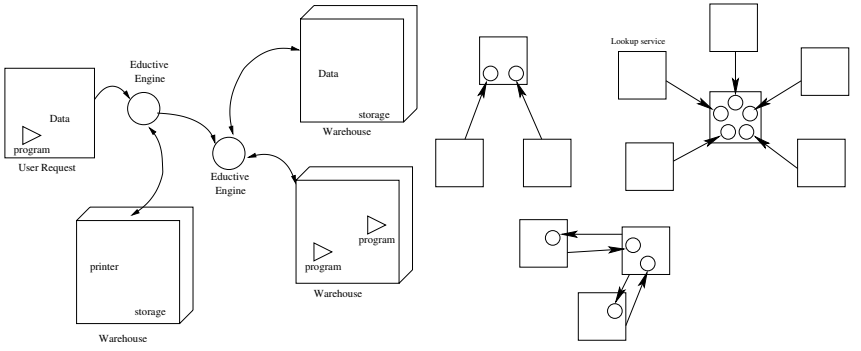
If we take a step backwards now, we can look at our framework in an entire different way. We can use the federation of busdevices as a simulator for our home automation system. We simply have to disconnect it from the physical bus and replace our buscommunicator service with a dummy one that simply acts as a loopback device. In this state, we can apply changes to the system and do some tests on it by letting the components send telegrams to each other (they get delivered thanks to the loopback device), without affecting the configuration of our home automation. Once the new configuration looks fine in the simulation, we can apply the adjustments to the memory of the real EIB devices.

4 Jini Related Work

Choosing Jini was straightforward for us, because we had all the necesary Java related know-how and the Jini framework was ready to use. There are, however, other distributed middleware solutions available. One of them, also using Java as programming environment, is the Web Operating System[4], or WOS in short. Although the system is still under development and not completely ready to use, it is interesting to have a look at it, and compare it with Jini.

The WOS begins with the concept of an operating system that offers resources to its users. Whereas a standard operating system keeps track of all

resources available on a machine, a distributed operating system can not keep track on all the available resources any longer, since it runs on machines scattered all over the network. To solve this, some kind of dynamism was built in the framework, based on a demand driven technique called eduction[7]. The eduction engines are the kernels of the WOS, which reactively answer to the requests of users or other eductive engines. It is very important to notice that the WOS is a versioned system. Different versions of WOS services can simultaneously be available on the network. This is a important conceptual difference with Jini as it influences the used discovery protocol and the way to interact with services. The most important consequence is that no interfaces are agreed upon in advance, which define the cooperation between clients and services in Jini. We will compare now briefly sections, some features of the Web Operating System with Jini.



(a) WOS eductive engines (b) Jini lookup hierarchies

Fig. 5. WOS versus Jini hierarchies

The discovery protocol. Jini uses a simple multicast protocol for discovering new services. The WOS, on the other hand, utilizes a more complex two-level communication protocol: the WOS Request protocol (WOSRP[8]) and the WOS Protocol (WOSP[8]). The WOSRP is the version independent layer, which looks up WOS servers and compares its version to the others to find compatible ones. The WOSRP iteratively broadcasts messages beginning with the nearest networks to networks further away to build a view of its environment. Once discovery is completed, the more powerful WOSP protocol is then used between mutual compatible WOS nodes. Jini on the other hand, only lets each service register itself with the local lookup in order to get found by its clients.

Lookup. Lookup is the term used for searching services based on certain conditions. The WOS keeps for this purpose references to its services in resource sets,

which are found via eductive engines. These eductive engines query the resource sets in their immediate environment. When the eductive engine can't handle the request by itself, it forwards the request to other eductive engines, until the resource finally gets found. Figure 5(a) shows this propagation of requests.

Similar constructions can be seen in Jini federations, although they are not automized as it is the case with eductive engines. Jini services usually register only themselves with the local lookup service. This way, each lookup service only manages the small group of services on his own network. Since lookup services are also common Jini services, they can register themselves with each other to build hierarchies of lookup services. Some hierarchies can be seen on figure 5(b). In such configurations, each lookup service typically maintains one Jini federation. A user who wants to use some service on another federation can then browse the hierarchy until he finds it.

Method calls. Both in Jini as in the WOS, Java RMI is used for the interaction between clients and services. Executing a remote call, nevertheless, is only possible when the method's signature is known in advance. Normally this signature is agreed upon in the service's interfaces, which is the case with Jini, but not in the WOS. The WOS separates methods in 'inputs' and 'outputs'. Inputs can be matched to outputs when the number and the type of their arguments match. The operation name doesn't have to match because it is called via an indirect message, which allows dynamic binding of inputs with outputs.

The people who designed the WOS had to use this kind of approach, because they think that standardizing interfaces causes too many troubles in a dynamic and continuously evolving environment. People often disagree when standarizing these interfaces, which hinders a smooth and fast evolution. That is why the WOS allows different versions of a service at the same time on the network.

Even the best ideas have drawbacks. Throwing away interfaces between the two parties (those who design services and those who use them) has major consequences on the development of software. Interfaces should include their specification, which forms the basis for contractual programming. It is impossible to keep the pre- and postconditions[9] of a method call in mind when only its signature is known. A programmer who does not write pre- and postconditions will sooner or later make programming errors that are hard to debug, which is a big disadvantage of a system with a broad version space like the WOS.

5 Conclusion

This paper gave an introduction to the EIB home automation system and the Jini architecture. The focus went to the integration of EIB components into a distributed environment with Jini. The result is a **dynamic** framework that can reflect the functionalities of the EIB devices **automatically** in a generated software model, which paves the road to incorporate domotics in an ubiquitous computing environment. Users or other software can now easily control and

reconfigure the devices. This can be done, when desired, through easy to use graphical interfaces that are integrated in the system.

We learned that the Jini framework is ready to use, but has not yet an easy high-level API, which would make it more mature. The help of EIB component manufacturers would services that are molded to fit with their application program. This would complete our framework.

Future work might consist in further developing the libraries for the service controller and in evaluating the WOS for use as alternative for Jini.

References

1. European Installation Bus Association: EIBA Handbook Series. EIBA, Brussels, 1999. http://www.eiba.com.
2. JNet Systems: EIBLETs, 2000. http://jnetsystems.com, http://www.eiblet.com
3. Kastner, W., Krugel, C.: Jini connectivity for EIB home and building networks - from design to implementation. In EIB Scientific Conference, Germany (1999)
4. Kropf P.: Overview of the WOS project. Advanced Simulation Technologies Conferences (ASTC 1999), San Diego, CA, USA (1999).
5. Sun Microsystems: Jini architectural overview. Technical White Paper, 1999. http://www.sun.com/jini/whitepapers.architecture.html
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, Massachusetts (1994)
7. Plaice, J., Lamine, S.: Eduction: A general model for computing. In Intensional Programming II (Singapore), World Scientific (1997)
8. Babin, G., Kropf, H., Unger, H.: A TwoLevel Communication Protocol for a Web Operating System (WOS TM). In: IEEE Euromicro Workshop on Network Computing (V aster as, Sweden), (1998) 939–944.
9. Meyer, B.: Object-Oriented Software Construction, Second Edition. Prentice Hall, Englewood Cliffs (NJ), USA (1997)

Author Index