TUCS

Lu Yan

Systematic Design of Ubiquitous Systems

Turku Centre *for* Computer Science

TUCS Dissertations
No 70, December 2005

Systematic Design of Ubiquitous Systems

Lu Yan

To be presented, with the permission of the Faculty of Chemical Engineering of the Åbo Akademi University, for public criticism in Auditorium 3102 at the Department of Computer Science at Åbo Akademi University, on December 13th, 2005, at 10 morning.

Department of Computer Science
Åbo Akademi University

2005

Supervisor

Professor Kaisa Sere
Department of Computer Science
Åbo Akademi University
Lemminkäisenkatu 14
FIN-20520 Turku
Finland

Reviewers

Professor Ralf Steinmetz
KOM - Multimedia Communications Lab
Technische Universität Darmstadt
D-64283 Darmstadt
Germany

Professor Nahid Shahmehri
Department of Computer and Information Science (IDA)
Linköpings Universitet
S-58183 Linköping
Sweden

Opponent

Professor Antti Ylä-Jääski
Helsinki University of Technology
P.O. Box 5400
FIN-02015 HUT
Finland

ISBN 951-29-4020-5
ISSN 1239-1883

Acknowledgements

I owe my deepest gratitude to my supervisor, Professor Kaisa Sere, who is a wonderful scholar and a demanding role model. By giving me the freedom to choose *what*, *when* and *why*, she often shows me *how*. Despite her busy schedule, she has always managed to find time for discussions on various aspects of my research. Without her continuous encouragement and friendly support combined with invaluable expert advice, this dissertation would have never been finished.

Professor Ralf Steinmetz from Technische Universität Darmstadt, Germany and Professor Nahid Shahmehri from Linköpings Universitet, Sweden have kindly agreed to review this dissertation and provided valuable feedbacks. Professor Antti Ylä-Jääski from Helsinki University of Technology, Finland has kindly accepted the role of opponent for the public defense. I wish to thank them all for their time and effort.

I would like to thank the Department of Computer Science at Åbo Akademi University and Turku Centre for Computer Science (TUCS) for generous financial support and excellent work environment provided during my studies. I would like to express my gratitude to the faculty, staff, former members, colleagues and friends here, for your help, assistance and cooperation, especially Ralph-Johan Back, Johan Lilius, Xinrong Zhou, Linas Laibinis, Luigia Petre, Marina Waldén, Elena Troubitsyna, Mauno Rönkkö, Rimvydas Ruksenas, Patrick Sibelius, Zheng Liang, Juha Plosila, Mats Neovius, Mats Aspnäs, Paul Lindholm, Jan Westerholm, Linda Grandell, Kim Solin, Marcus Alanen, Pontus Boström, Orieta Celiku, Fredrik Degerlund, Dubravka Ilic, Sebastien Lafond, Chang Li, Robert Löfman, Sofia Nygård, Cristina Seceleanu (Cerschi), Qaisar Malik, Leonidas Tsiopoulos, Dan Österberg, Jincheng Ni, Nayyar Iqbal, Guangcheng Niu, and Moisés Ferrer Serra. In particular, I would like to thank my first scientific advisor here, Professor Joakim von Wright, for providing guidance during my Master studies.

Finally, I would like to thank my parents. Your love, understanding, and encouragement have helped me complete this work. This dissertation is dedicated to you.

Turku, November 2005
Lu Yan

List of Original Publications

I. L. Yan, K. Sere. Stepwise Development of Peer-to-Peer Systems. In *Proceedings of the 6th International Workshop in Formal Methods (IWFM'03),* July 2003. British Computer Society Press.

II. L. Yan. Via Firewalls. In *Proceedings of the 3rd International Conference on Grid and Cooperative Computing (GCC'04)*, October 2004. Lecture Notes in Computer Science 3252, Springer-Verlag.

III. L. Yan, M. F. Serra, G. Niu, X. Zhou, K. Sere. SkyMin: A Massive Peer-to-Peer Storage System. In *Proceedings of the 3rd International Conference on Grid and Cooperative Computing (GCC'04),* October 2004. Lecture Notes in Computer Science 3251, Springer-Verlag.

IV. L. Yan, J. Ni. Building a Formal Framework for Mobile Ad Hoc Computing. In *Proceedings of the International Conference on Computational Science (ICCS'04)*, June 2004. Lecture Notes in Computer Science 3036, Springer-Verlag.

V. L. Yan, K. Sere, X. Zhou, J. Pang. Towards an Integrated Architecture for Peer-to-Peer and Ad Hoc Overlay Network Applications. In *Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04),* May 2004. IEEE Computer Society Press.

   Shorten version available as: L. Yan. MIN: Middleware for Network-Centric Ubiquitous Systems. In *IEEE Pervasive Computing*, Vol. 3, No. 3, July - September 2004.

VI. L. Yan. Performance Evaluation and Modeling of Peer-to-Peer Systems over Mobile Ad hoc Networks. Submitted to *Performance Evaluation*.

   Previous version available as: L. Yan. Performance Evaluation and Modeling of Peer-to-Peer Systems over Mobile Ad hoc Networks. TUCS Technical Reports, No. 678, Turku Centre for Computer Science, Finland, 2005.

   Shorten version available as: L. Yan. Performance Modeling of Mobile P2P Systems. In *Proceedings of the International Conference on Computer Networks and Mobile Computing (ICCNMC'05)*, August 2005. Lecture Notes in Computer Science 3619, Springer-Verlag.

.

VII. L. Yan, K. Sere. Formal Context-Aware Programming in Mobile Environments. Submitted to *Scientific Programming*.

Previous version available as: L. Yan, K. Sere. A Formalism for Context-Aware Mobile Computing. In *Proceedings of the 3rd International Symposium on Parallel and Distributed Computing and the 3rd International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04),* July 2004. IEEE Computer Society Press.

VIII. Z. Liang, L. Yan, J. Plosila, K. Sere. Implementing an Asynchronous Java Accelerator for Ubiquitous Computing. Submitted to *Journal of Embedded Computing*.

Previous version available as: L. Yan, Z. Liang. Accelerating Java for Ubiquitous Devices. In *Proceedings of the 4th International Conference on Computer and Information Technology (CIT'04),* September 2004. IEEE Computer Society Press.

IX. L. Yan. Formal Verification of a Ubiquitous Hardware Component. In *Post-Proceedings of the First International Conference on Embedded Software and System (ICESS'04)*, 2005. Lecture Notes in Computer Science 3605, Springer-Verlag.

Extended version available as: L. Yan. Formal Verification of a Ubiquitous Hardware Component. TUCS Technical Reports, No. 637, Turku Centre for Computer Science, Finland, 2004.

Contents

Part I: Research Summary vii

Introduction 1
Software-Intensive Work 4
Hardware-Intensive Work 12
Related Work 15
Final Words 23
Bibliography 24

Part II: Publication Reprints 33

PART I:
RESEARCH SUMMARY

1 Introduction

Ever since the last decade, we have been witnessing an age of ubiquity where our home and workplace are being transformed by distributed computing. Computers are moving off the desktop and into every part of our lives - our cars, our living rooms, and even our coat pockets. At the same time, many other devices, from televisions and music systems to home appliances, are themselves turning into computers, becoming as intelligent and connected as the PCs of today.

A decade ago, Mark Weiser[1] brought forth the famous comment “The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.”, which is *de facto* the origin of ubiquitous computing [1, 2, 3]. The essence is the gracefully integrated software and hardware to support and ease daily activities of human society. From then on, various projects were launched in universities and companies around the world on ubiquitous computing, including European Community’s ambient intelligence (AmI) initiative, Georgia Tech’s Aware Home, Inria’s Smart Office, Stanford’s iRoom, Cisco’s Internet Home, Essex’s Intelligent Inhabited Environments, HP’s Cool Town, ATR’s Creative Space, CMU’s Aura, Xerox’s Smart Media Spaces, IBM’s DreamSpace, KTH’s comHOME, Microsoft’s EasyLiving, MIT’s Oxygen, Philips’ Home of the Future, UW CSE’s Portolano, Intel’s Proactive Health, UF’s Assistive Smart House, Keio’s SSLab, Cambridge’s TIME, etc.

Ubiquitous systems touch on a broad array of disciplines. Though the above projects address various aspects of ubiquitous systems, the design methodology of these systems has not received enough attention. Nonetheless, all of these smart objects and their applications are to be implemented into our everyday environments. Those systems should fulfill critical requirements such as reliability, availability, safety, security, etc. To meet this objective, the systematic design methodology is needed and should be applied to various stages of ubiquitous systems design flow, such as specification, refinement, verification, etc.

In this thesis, efforts towards systematic design of ubiquitous systems are elaborated. As we were trained with formal methods, several approaches in this thesis are tailored in the style of formal methods. A formal method for system development is an approach based on rigorous mathematical foundation, which aims to establish that the derived implementation adhere correctly to its given specification. The advantage of using formal methods in system development process lies in the precise modeling capability and the further verification and refinement support, which enables a stepwise development from specification to implementation [4].

Informal methods, such as UML modeling, network simulation and testing, hardware software co-design, are also used in some approaches in this thesis. We argue that there is no one method that fits all, and a synergistic interweaving of

[1] Father of “ubiquitous computing”.

formal and informal methods will be the better way to our system design process, where the strength of formal and informal methods complements each other [5, 6]. Therefore, in this thesis, we try to keep a good balance of formal and informal methods, taking the advantages from both.

The thesis is organized as an introductory part and a collection of papers. The introductory part consists of five sections: Section II concentrates on the software infrastructure - on building and evaluating ubiquitous system software. Section III is dedicated to hardware infrastructure - on building and accelerating ubiquitous system hardware. Section IV discusses a number of related works and section V concludes the thesis with final words. The collection of papers is listed below:

X. L. Yan, K. Sere. Stepwise Development of Peer-to-Peer Systems. In *Proceedings of the 6th International Workshop in Formal Methods (IWFM'03),* July 2003. British Computer Society Press.

XI. L. Yan. Via Firewalls. In *Proceedings of the 3rd International Conference on Grid and Cooperative Computing (GCC'04)*, October 2004. Lecture Notes in Computer Science 3252, Springer-Verlag.

XII. L. Yan, M. F. Serra, G. Niu, X. Zhou, K. Sere. SkyMin: A Massive Peer-to-Peer Storage System. In *Proceedings of the 3rd International Conference on Grid and Cooperative Computing (GCC'04),* October 2004. Lecture Notes in Computer Science 3251, Springer-Verlag.

XIII. L. Yan, J. Ni. Building a Formal Framework for Mobile Ad Hoc Computing. In *Proceedings of the International Conference on Computational Science (ICCS'04),* June 2004. Lecture Notes in Computer Science 3036, Springer-Verlag.

XIV. L. Yan, K. Sere, X. Zhou, J. Pang. Towards an Integrated Architecture for Peer-to-Peer and Ad Hoc Overlay Network Applications. In *Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04),* May 2004. IEEE Computer Society Press.

Shorten version available as: L. Yan. MIN: Middleware for Network-Centric Ubiquitous Systems. In *IEEE Pervasive Computing*, Vol. 3, No. 3, July - September 2004.

XV. L. Yan. Performance Evaluation and Modeling of Peer-to-Peer Systems over Mobile Ad hoc Networks. Submitted to *Performance Evaluation*.

Previous version available as: L. Yan. Performance Evaluation and Modeling of Peer-to-Peer Systems over Mobile Ad hoc Networks. TUCS Technical Reports, No. 678, Turku Centre for Computer Science, Finland, 2005.

Shorten version available as: L. Yan. Performance Modeling of Mobile P2P Systems. In *Proceedings of the International Conference on Computer Networks and Mobile Computing (ICCNMC'05)*, August 2005. Lecture Notes in Computer Science 3619, Springer-Verlag.

.

XVI. L. Yan, K. Sere. Formal Context-Aware Programming in Mobile Environments. Submitted to *Scientific Programming*.

Previous version available as: L. Yan, K. Sere. A Formalism for Context-Aware Mobile Computing. In *Proceedings of the 3rd International Symposium on Parallel and Distributed Computing and the 3rd International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04),* July 2004. IEEE Computer Society Press.

XVII. Z. Liang, L. Yan, J. Plosila, K. Sere. Implementing an Asynchronous Java Accelerator for Ubiquitous Computing. Submitted to *Journal of Embedded Computing*.

Previous version available as: L. Yan, Z. Liang. Accelerating Java for Ubiquitous Devices. In *Proceedings of the 4th International Conference on Computer and Information Technology (CIT'04),* September 2004. IEEE Computer Society Press.

XVIII. L. Yan. Formal Verification of a Ubiquitous Hardware Component. In *Post-Proceedings of the First International Conference on Embedded Software and System (ICESS'04)*, 2005. Lecture Notes in Computer Science 3605, Springer-Verlag.

Extended version available as: L. Yan. Formal Verification of a Ubiquitous Hardware Component. TUCS Technical Reports, No. 637, Turku Centre for Computer Science, Finland, 2004.

We briefly summarize some major contributions and give a map to where in the thesis those contributions appear as follows. The complete and detailed contributions are available in the summaries of the papers in section II and section III.

- Integrated formal and informal specification: Paper I, Paper II, Paper III, and Paper IV.
- Integrated P2P and MANET network architecture: Paper V and Paper VI.
- Formal approach to context-awareness: Paper VII.
- Hardware design and formal verification: Paper VIII and Paper IX.

2 Software-Intensive Work

This section concentrates on software infrastructure - on building and evaluating system software. The papers that have been published about the work in this section are **Paper I - Paper VII**, many of which are on software architectures in ubiquitous networking environments, because a critical issue in the design and construction of any complex software system is its architecture [149]. In the following paragraphs, we present a short summary of these papers.

We started the research on ubiquitous software from neo-distributed systems, more specifically, peer-to-peer systems. A peer-to-peer system consists of a distributed set of peer-to-peer nodes. Every node can act both as a server and as a client. The most characteristic features of a peer-to-peer network are that every node is able to (a) make information available for distribution to other nodes, (b) establish a connection to any other node in the network and (c) have access to the information that the other nodes of the network provide. These systems give rise to application-level virtual networks with routing mechanisms of their own [7].

Peer-to-peer systems have recently become a very active research area due to the popularity and widespread use of peer-to-peer systems today and their potential uses in future applications. Peer-to-peer systems are characterized by a very large number of autonomous computing nodes (the peers), which pool together their resources and rely on each other for data and service. In this way, they offer many benefits: adaptation, self-organization, load-balancing, fault-tolerance, availability through massive replication, and the ability to pool together and harness large amounts of resource. These benefits make peer-to-peer networking a promising candidate towards ubiquitous networking: after establishing basic connectivity, ubiquitous devices could self-organize and cooperatively provide network services that are normally provided by infrastructure servers, which are often absent in ubiquitous computing environments.

Paper I: Stepwise Development of Peer-to-Peer Systems is a case study of stepwise development of a Gnutella-like [8] peer-to-peer system where we first study the peer-to-peer networking paradigm. The reason for choosing Gnutella[2] is that it is the most typical pure peer-to-peer system and also the most studied one: the Gnutella model enables file sharing without using servers. Unlike a centralized server network, the Gnutella network does not use a central server to keep track of all user files.

At that time this paper was written, the peer-to-peer networking applications, especially file sharing ones, were booming via the Internet. One search in Google would return hundreds of peer-to-peer applications, developed academically or commercially. After using and analyzing various peer-to-peer clients on different platforms, we identified two common problems of the clients at that time being:

[2] Gnutella is a decentralized peer-to-peer file-sharing model developed in 2000 by Nullsoft, AOL subsidiary and the company that created WinAMP.

reliability (Most clients fail to provide satisfactory download service and some buggy ones even bring down the system during our test) and extendability (Most clients are implemented in a way that adding new services or functionalities results to many modifications to the original specifications). These weak points motivated us to conduct a case study on rigorous development of these kinds of applications.

As proposed in this paper, an attractive strategy to solve the first problem is to use formal methods in designing peer-to-peer systems. Formal methods can help with reliability by minimizing errors in designing peer-to-peer systems. To improve extendability, we introduce a modular and object-oriented architecture for peer-to-peer systems, which favors a reusable, composable and extendable design. Those are the main contributions of this paper. Another novel point in this paper is that it is the first initiative, as far as is known, to apply formal methods to the design process of peer-to-peer systems and the result turns out to be positive.

The final specification in this paper is easily implemented in OO-languages. Since one of our design goals is to build a reliable peer-to-peer system for all platforms with possible future extensions to mobile systems such as PDAs and smart phones, we eventually implemented the formal specification in Java [9], partly because it is the most popular cross-platform language today. Moreover, the code is released under GNU public open license to encourage contributions from the open source community.

Paper II: Via Firewalls is a step further into the previous approach. When implementing our system in Java, we realized that many networks today are actually behind firewalls. In peer-to-peer networking settings, firewall-protected peers may have to communicate with peers outside the firewall. In this case, a solution should be made to create communication schemes that overcome the obstacles placed by the firewalls to provide universal connectivity throughout the network. This problem occurred in real-world experiments and deployments motivated us to conduct a study of firewalls in peer-to-peer networking and achieve a way to traverse firewalls.

Firewalls usually examine the packets of information sent at the transport level to determine whether a particular packet should be blocked. Each packet is either forwarded or blocked based on a set of rules defined by the firewall administrator.

With packet-filtering rules, firewalls can easily track the direction in which a TCP connection is initiated. A common configuration for these firewalls is to allow all connections initiated by computers inside the firewall, and restrict all connections for computers outside the firewall. For example, firewall rules might specify that users can browse from their computers to a web server on the Internet, but an outside user on the Internet cannot browse to the protected user's computer. In order to traverse these kinds of firewalls, we extended our previous specification and introduced a new descriptor *Push* and routing rules, which would effectively bridge the communication between one peer inside a uni-directional firewall and the other peer outside.

Other kinds of common firewalls are port-blocking ones, which usually do not grant long-time and trusted privileges to ports and protocols other than *80/443* and *http/https* respectively. For example, port *21* (standard *ftp* access) and port *23* (standard *telnet* access) are usually blocked and applications are denied network traffic through these ports. In this case, *http* would be the permitted entry to the network. Using *http* protocol, for a peer outside to communicate with another peer inside this firewall, it has to pretend that it is an *http* server, serving *www* documents. In other words, it has to mimic an *httpd* program. In order to traverse these kinds of firewalls, we extended our previous architecture and introduced a new layer as *proxy*, which would act as a tunnel between peer and the Internet with the above ideas.

As an extension of the previous paper, the main contribution of this paper is the refined specification and software architecture to solve the firewall problems in real-world peer-to-peer experiments and deployments.

The increasing demand for massive storage systems has spawned an urge for a large-scale storage solution with scalability, high availability, persistence and security. Nowadays, the Internet has become less expensive and widespread, which makes it possible to build an economical massive storage solution over the Internet. It is possible to build a ubiquitous and global persistent data storage solution designed to scale to billions of users and devices [12]. Since a major merit of the peer-to-peer overlay is the enabling technologies that empower the leaf-nodes underneath [59], one natural application for the overlay peer-to-peer settings is ubiquitous and global storage.

Paper III: SkyMin: A Massive Peer-to-Peer Storage System is proposed under the above context. The aim is to design a large-scale, Internet-based storage system providing scalability, high availability, persistence and security.

Our approach to constructing this massive storage file system is to implement a layer on top of existing heterogeneous file systems. The architecture is simply described as follows. It consists of many FS (File Server) and several NS (Name Server). NS is the control center and FS is the storage unit. Every node in this system serves as an access point for users. Nodes are not trusted; they may join the system at any time and may silently leave the system without warning. Yet, we hope the system is still able to provide strong assurance, efficient storage access, load balancing and scalability.

This paper documents the whole design process of the development of *SkyMin* prototype. We start with the functional requirements of our system from both users' and system's view, and then we address the non-functional requirements with necessary tradeoffs, since some of them are contradictory in nature.

The system architecture consists of one (or more) central server and many peers which are connected to the Internet. The server works as a control point to grant the peer's access to the storage system, to manage the indexes of the peers' resources (files can be accessed by other peers who are members of the whole storage system), etc. On the other hand, every joined peer contributes part of their local storage space to the system, and the peer can save and retrieve files to/from the system.

For the software architecture, we specified the static design with UML class diagram and the dynamic design with UML sequence diagram. Following the exact specification, our prototype is implemented in Java with support of MySQL.

The main contribution of this paper is the detailed specification of our prototype. It describes detailed accounts of a completed peer-to-peer software-system project which can serve as a *how-to-do-it* model for future work in the same field. Although this prototype is a working subset of the initial goal presented above and several design elements still need fine-tuning, it is already expressive enough to support several interesting scenarios, including a groupware for small-scale information storage in a secure way, and a distributed data center for smart home appliances.

A mobile ad hoc network (MANET) is a self-organizing and rapidly deployable network in which neither a wired backbone nor a centralized control exists. The network nodes communicate with one another over wireless medium. Nodes can only directly communicate with their neighbors. Therefore, distant nodes must communicate with one another in a multi-hop fashion. MANET is adaptable to highly dynamic topology resulted from mobility of network nodes and does not require any fixed infrastructure such as base stations, therefore, it is an attractive networking option for connecting mobile devices quickly and spontaneously. Such merits as self-organizing and rapidly deployable make MANET also a promising candidate for ubiquitous networking.

Paper IV: Building a Formal Framework for Mobile Ad Hoc Computing is formal framework from the application developer's view. After carefully studying the typical software pattern for MANET applications, we defined a layered software architecture framework with three key components, *Network Management*, *Awareness* and *Interaction*, which buildup an intermediate layer between the software application layer and the ad hoc networking layer.

This paper specifies our system components with the B method [13], and models the interactions and message communications between the components with UML diagrams. In our framework, the three key components model different aspects of mobile ad hoc computing. Due to the mobility of nodes in MANET, our system is focused on the awareness of the environments, the connections between nodes and the communication of nodes. The routing issues are also considered when nodes are communicating with each other.

Contributions from this paper are the proposed software architecture and the detailed specification of each component. It forms a formal framework to enable applications to be developed based on the three key components, which are supposed to be executed arbitrarily in MANET. Two case studies were done in order to testify the feasibility of our approach [14]: one is a peer-to-peer chatting application *BlueChat* running on Bluetooth devices and the other is an experiment of ad hoc network establishment and routing testing using LAN access profile of Bluetooth [15].

Based on the investigations of P2P and MANET, which were already intensively discussed in the above paragraphs, we found out that P2P and MANET share some key characteristics: self-organization and decentralization, and both need to solve the same fundamental problem: connectivity.

Previously, the P2P and MANET research communities have been working largely in isolation, while facing many common issues such as self-organization and decentralization. We argue that it is a promising research direction to bring the two communities together to merge the techniques used in the two areas and perhaps discover unified tricks for the convergence of the two overlay network technologies.

Therefore, we conducted a study for the convergence of the two overlay network technologies and proposed an evolving architecture towards integrating the two technologies in building overlay network applications.

Paper V: Towards an Integrated Architecture for Peer-to-Peer and Ad Hoc Overlay Network Applications discussed the similarities between the two overlay networks such as (1) Dynamic topology: A node in P2P and MANET may join or leave the network at any time and the position of a node in MANET is changing arbitrarily, which leads to no constant routes for any nodes. Both networks have a dynamically changing network topology. (2) Hop connection: Connections in P2P and MANET are established via exchanging beacon messages only between neighbor nodes. A single hop connection in P2P is typically via TCP links without physical limits, while a single hop in MANET is via wireless links which are usually limited by the radio transmission range. (3) Routing protocol: Both P2P and MANET routing protocols have to deal with dynamic network topologies due to membership changes or mobility. Typically, a host in P2P and MANET also serves as a router, and employs some flooding-based routing protocols. These common characteristics shared by P2P and MANET also lead to the same fundamental challenge, that is, how to provide connectivity in a completely decentralized environment.

MIN architecture is the proposal of future network architecture under the context: Today's networks are dependent on wired or wireless infrastructure. This dependence renders the networks vulnerable to disasters and attacks against the fixed infrastructure that supports them. Disasters such as floods and earthquakes, as well as wars and terror strikes, can damage or shutdown the whole network. Thus a state-of-the-art research direction of nowadays network is on connectivity. A network architecture that satisfies the above scenario will be radically different from the current existing network architectures since it cannot rely on a fix infrastructure and dedicated servers.

Recent work on P2P overlay networks offers a self-organizing substrate for decentralized network applications. Our general approach is to build a structured P2P overlay with existing technologies upon the basic connectivity provided by MANET in the absence of a dedicated server infrastructure. However, an important challenge is that existing P2P overlay protocols were designed for the Internet, which is a very different environment than MANET. The unique characteristics of this emerging class of networks calls for novel architectures. We present the key challenges as a set of research problems: (1) Self-organizing infrastructure: Wired networks rely on a fixed infrastructure consisting of routers and DHCP and DNS servers. Any damage or interfere of the server will probably make the whole network out of service. Emerging P2P technologies promise to support self-organizing infrastructure, but these technologies are not directly applicable to the ad hoc wireless environments, because they are originally designed for the Internet with constantly stationary nodes, where as nodes are arbitrarily moving in MANET. (2) Decentralized service: Existing networks depends on dedicated servers providing centralized basic network services such as naming, authentication and timing etc. For instance, conventionally there are DHCP and DNS services in a typical network, while supporting these kinds of critical network services is beyond the capability of existing P2P networks. Our approach is to build foundations from P2P systems, but take advantages of the hierarchical overlay structure contributed by MANET to provide decentralized network services. (3) Integrated routing: Integrating a P2P routing protocol into a MANET protocol is difficult. P2P overlays in the Internet rely on the IP routing mechanism which is actually application-level routing, while such kind of routing is usually carried out in link-level in MANET. Although typical flooding and multi-hop routing protocols in MANET are peer-to-peer in natural, P2P routing protocols are not directly applicable in MANET.

Based on the above analysis, we propose an evolving architecture which is able to provide network connectivity in a decentralized fashion and use self-organizing infrastructures to improve availability of today’s network. In this architecture, ad hoc wireless networks can be combined with infrastructure-based networks through ad hoc communications between them. Once basic connectivity is established, hosts could self-organize and cooperatively provide network services that are normally provided by infrastructure servers.

The main contribution of this paper is that it proposes a novel architecture, integrating P2P and MANET technologies together, to reduce the dependence of networking on wired and wireless infrastructure, thus extending the reachability of nowadays networks and increasing their resilience to disasters and attacks. Another contribution of this work is that it is the first architecture-centric approach, as far as known, for the construction of overlay network applications that allows us to define a unified networking environment, taking advantages from both P2P and MANET technologies.

Paper VI: Performance Evaluation and Modeling of Peer-to-Peer Systems over Mobile Ad hoc Networks is a step further into the proposed architecture. With the experiences and lessons learned from our summer projects, we identified more technical points on P2P and MANET: (a) P2P is generally referred to the application layer, but MANET is generally referred to the network layer, which is a lower layer concerning network access issues. Therefore, the immediate result of this layer partition reflects the difference of the packet transmission methods between P2P and MANET: the P2P overlay is a unicast network with virtual broadcast consisting of numerous single unicast packets; while the MANET overlay always performs physical broadcasting. (b) Peers in P2P overlay are usually referred to static nodes though no priori knowledge of arriving and departing is assumed, but peers in MANET are usually referred to mobile nodes since connections are usually constrained by physical factors such as limited battery energy, bandwidth, computing power, etc.

We then studied the performance issues of P2P systems over MANET from users' point of view since it makes greater impact on the design decisions of such kinds of systems for mobile operators, value-added service providers and application developers. Specifically, we want to answer the following questions: (1) How can we perform an efficient search in mobile P2P systems? (2) and what is the performance experience when many users try to retrieve data with parallel downloading scheme?

To answer the first question, the routing protocols and route discovery efficiency of different settings for the peer-to-peer overlay and underlying mobile ad hoc network were further investigated. There are many routing protocols in P2P networks and MANET respectively, but all of them fall into two basic categories: broadcast-like and DHT-like. Therefore, we conducted a survey of integrating these protocols in different ways according to categories. As a result, we show that the cross-layer approach performs better than separating the overlay from the access networks.

To answer the second question, we believe that rigorous analytical models are definitely needed, which capture the relation between key system parameters and performance metrics. After characterizing the variability of the system by taking some preliminary assumptions, we then present a performance model which captures most facets of mobile peer-to-peer systems. The novelty of our model comes from the classification of nodes: *contributor*, *leech*, *seed*, and we believe it is reasonable to have *leech* (pure downloader) in the model since the realistic implication of this type may be physically constrained mobile devices (e.g. cellular phones with limited bandwidth or associated with too expensive data transmission charge). We then briefly discussed three analytical examples on applying this model to capture the behavior of the system in steady states.

The main contribution of this paper lies in the answers to the above two questions. In other words, we conducted a survey of integrating P2P and MANET overlays in different ways according to categories and presented a performance model which captures most facets of mobile peer-to-peer systems. We hope our results would potentially provide useful guidelines for mobile operators, value-added service providers and application developers to design and dimension mobile peer-to-peer systems, and as a foundation for the architecture proposed in the previous paper.

Along the road of the proposed network architecture, one natural direction is the ubiquitous application development, since nowadays, mobile computing devices, such as notebooks, mobile phones, PDAs and digital cameras have gained wide-spread popularity. Although these devices and their networking capabilities are becoming increasingly powerful, the design of mobile applications will continue to be constrained by physical limitations. Mobile devices will continue to be battery-dependent and users are reluctant to carry heavyweight devices. Networking capabilities will continue to be based on communication with base-stations, with fluctuations in bandwidth depending on physical location.

In order to provide acceptable QoS to the users, applications have to be context-aware [16] and able to adapt to context changes, such as variations in network bandwidth, exhaustion of battery power or reachability of services on other devices. This would require application developers, for example, to manage and process useful context information from the user's surroundings, and adapt to it accordingly.

Paper VII: Formal Context-Aware Programming in Mobile Environments is proposed in order to ease the development of context-aware applications and ensure the correctness of their designs. In this paper, we extend the classical actions systems formalism with context information and define the context-aware action systems that provide a systematic method for managing and processing context information. The meaning of context-aware action systems is defined in terms of classical action systems [17], so that the properties of context-aware action systems can be proved using standard action systems proof techniques. Moreover, action systems are intended to be developed in a stepwise manner within an associated refinement calculus [18]. Hence, the development and reasoning about context-aware action systems can be carried out within this calculus ensuring the correctness of derived mobile applications.

Compared to the classical approach, the novelty is the interpretation of context. Hence, we can say that context-aware action systems form a subset of action systems. After a brief introduction to our interpretation of context and some essential notions and properties of this formalism, we present a context-aware scenario as an example to show how this context-aware action systems framework can be effectively used to model context-aware services for mobile applications. The ideas behind this scenario are rooted in the notion that mobile application development could be simplified if the retrieval and maintenance of context information were to be delegated to the software support infrastructure without loss of flexibility and generality, which could be, for instance, a middleware developed from our previously proposed network architecture.

The novel contribution of this paper is the formal design and formalism that facilitate the development of context-aware applications. In particular, we have described a formal approach to context-aware mobile computing: we offer the context-aware action systems framework, which provides a systematic method for managing and processing context information, defined on a subset of the classical action systems.

Ubiquitous computing poses a great impact on how future software architectures will be practiced. First, we need architectures suitable for extreme resource-constrained systems. Second, architectures for these systems have to be reconfigurable. Third, there is need for architectures that manage and handle mobility more automatically [149].

3 Hardware-Intensive Work

This section concentrates on hardware infrastructure - on building and accelerating ubiquitous system hardware. The papers that have been published about the work in this section are **Paper VIII** and **Paper IX**, and we present a short summary of them as follows.

We started our research on ubiquitous hardware with Java-enabled devices, partly because nowadays Java has become the most popular and portable language for its *write once, run anywhere* promise, which makes it ideal for developing applications for ubiquitous devices. The platform-independence is achieved via various Java Virtual Machines (JVMs): Java source codes are compiled into bytecode streams, which are to be executed in JVMs. Bytecode representations are in portable formats that allow programs, no matter small applets in embedded systems or large desktop applications, to run on as many platforms as possible. In this way the Java technology provides a common application interface regardless of the underlying platform, allowing a project to be moved to a new platform with a minimal or even completely without change, as long as the JVM exists for that platform.

While the Java technology comes with compromises as well. The most significant disadvantage of Java applications is on performance: the execution speed of Java bytecode in JVMs is not as fast as native code written in C/C++. The software mode execution engine is quite slow in interpreter or bigger code size in Just-in-time (JIT) compiler. A promising way of performance enhancement is to implement the JVM in hardware mode (i.e. silicon implementation) which can be optimized to deliver much better performance than software mode.

Paper VIII: Implementing an Asynchronous Java Accelerator for Ubiquitous Computing proposed a scheme of hardware accelerated JVM for existing ubiquitous systems. The accelerator is in ultra low power design, and can be integrated into existing processors and run time operation systems (RTOS). To meet the critical constraint of low power consumption of most ubiquitous consumer devices today, the accelerator is designed in asynchronous style.

Since one of our design goals is to support as many processors as possible, coprocessor mode is the best choice, which means that frequently used bytecode instructions will be executed by the Java accelerator. Another design goal of our Java accelerator is target for low end ubiquitous devices, where only one CPU accesses the memory and works as master module in system bus, and the computing capability of the system is quite limited.

The workflow of our Java accelerator is briefly described as follows: when the Java accelerator works with the main processor, the main processor needs to configure the Java accelerator in the beginning of every Java thread i.e. set new PC, stack address, segment offset, etc in the Java accelerator. After the beginning of Java bytecode stream, the main processor halts and the Java accelerator will fetch bytecode instruction and data from memory. When the Java accelerator encounters trap instructions that the accelerator can't implement, it sends interrupt and moves its control to the main processor, and then the main processor will access the stack cache in the Java accelerator as I/O operations. In bytecode trap handling, the main processor takes operands from Java stack, executes subroutine for trapped Java bytecode, and writes the result back into Java stack top. In case that the next bytecode should be executed in hardware, the main processor halts and updates PC in the Java accelerator to invoke it. Then the Java accelerator can continue to use the new result in stack. With this scheme, our Java accelerator can be integrated into most existing processors, and no bus arbiter is needed.

The main contribution of this paper is the proposed coprocessor scheme, which would be suitable for most existing Java-enabled ubiquitous consumer devices. Some preliminary simulation results of small applets prove that the scheme provides a new solution to accelerating Java for ubiquitous devices.

Nowadays, advances in silicon technology have made it possible to embed inexpensive processors, sensors, and actuators in just about anything and everything. As the result, ubiquitous hardware is becoming more and more powerful and capable, as well as complex. However, it becomes increasingly difficult to ensure these devices free of design errors. In most cases, exhaustive simulation of a medium size design is impossible and the correctness of the design cannot be assured. This is a serious problem in safety-critical applications, where a small design error may cause loss of life and extensive damage. Even in the case where safety is not the primary concern, a design error means costly and time-consuming rechecking in massive production lines.

A solution to the problem is to apply formal methods for verification of correctness of hardware designs - hardware verification. With this approach, the behavior of hardware is described mathematically, and formal proof is used to verify the intended behavior. The proofs can be very large and complex, so mechanical verification tools are often used to assist the verification.

Paper IX: Formal Verification of a Ubiquitous Hardware Component is our experiences report on formal verification in ubiquitous hardware design, via a comparative case study of the verification of a circuit design of a seven-segment LED display decoder.

A seven-segment LED display is comprised of seven light emitting diodes (LED). Input signals are applied to the input port of the seven-segment decoder, and the decoder translates them into ON/OFF status of the seven LEDs. Then, selected combinations of the LEDs are illuminated to display numeric digits and other symbols. Such kinds of hardware components are widely used in low-end ubiquitous consumer devices as the display unit, so it would be interesting to see how to make sure the circuit design of these components free of errors.

In this paper, we started with an overview on hardware verification methods, with the emphasis on approaches using higher order logic [19]. Afterwards, we selected two popular verification tools, HOL [20] and PVS [21], and illustrated with some well-understood, but nontrivial examples, then smoothly moved to our practical verification case study of a seven-segment LED display decoder circuit design. Since we have used both HOL and PVS, known as powerful proof tools, as the mechanical verification tools in our case, we conclude the paper with a comparative study of the two proof tools, and some possible future improvement suggestions for them according to our experiences.

As suggested, the main contribution of this paper lies in the experiences part. The treatment is detailed and is based on clear mathematical foundations. Moreover, it documented a full case study from requirement to specification to implementation to verification, which results from our personal practical experiences with tools and methods developed and used in both academic and industrial environments.

Throughout the history of computing, ubiquitous applications have been pushing the limits of computing power, especially in terms of real-time computation. On the other hand, recent advances in microelectronics and communication technology have allowed large numbers of portable devices to communicate both among themselves and with a wired or wireless infrastructure. All these above changes and trends bring us new opportunities as well as challenges in hardware design for the future ubiquitous devices. We conclude section III with our remark for the future ubiquitous devices:

Historically, ubiquitous systems have been highly engineered for a particular task, with no spontaneous interactions among devices. Recent advances in wireless communication and sensor and actuator technologies have given rise to a new genre of ubiquitous systems. This new genre is characterized as self-organizing, critically resource constrained, and network-centric. The fundamental change is communication: numerous small devices operating collectively, rather than as stand-alone devices, form a dynamic, multihop routing network that connects each device to more powerful networks and processing resources.

4 Related Work

In the literature several definitions of the term *ubiquitous* can be found [1, 2, 3, 22, 23, 24, 25, 26], but a detailed discussion towards the definition of ubiquitous is out of the scope of this thesis. The work [27, 28, 29, 30, 31, 32] had greatly influenced our understanding of *ubiquitous*, and we would like to interpret it as follows:

Ubiquitous Systems are about computer systems that are available everywhere. The ubiquitous computers are supposed to be implemented in all kinds of everyday things, and will communicate with the users of the system, and with each other. The purpose of ubiquitous computing is to help making its users lives easier, and to avoid information overload [27].

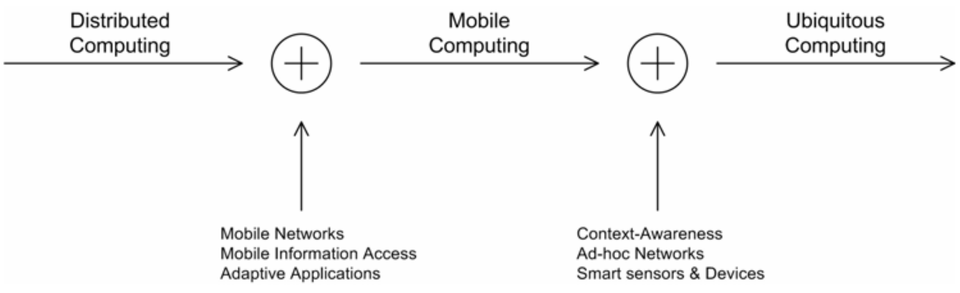


Figure 1. The Evolution Chain [28]

In other words, ubiquitous computing is a stepwise evolution from distributed computing and mobile computing in Figure 1. There are two issues that are important to make ubiquitous computing work: location and scale. To make a ubiquitous computer system, you will need a lot of computers located in a lot of places. These computers will need to be able to locate themselves, to know where they are. If a computer knows what room it is in, it can adapt its behavior in significant ways without requiring any artificial intelligence [1]. For ubiquitous computing to be implemented into our everyday environment, they will have to be almost everywhere. You will need to have hundreds of computers of different sizes in each room as the computers will replace all kinds of products, such as notebooks, thermostats, clocks, blackboards and bulletin boards. To make the system useful to its users, it also has to be available in all kinds of surroundings, both inside and outside of buildings. The technology that is required for ubiquitous computing comes in three parts [1]: (1) cheap, low-power computers that include equally convenient displays (2) a network that ties them all together (3) software systems implementing ubiquitous applications.

In the following paragraphs, we discuss a number of related works and some possible future directions.

If we date back to the computing history, P2P is as old as the Internet [57]. It attracts many attentions during recent years, partly because of the unexpected success of P2P paradigm in distributed computing [58]. The real prevalence of P2P paradigm over the Internet starts since 2001, due to the widely deployment of P2P file-sharing applications [59]. In the literature many survey on P2P paradigm can be found [60, 61, 62, 63, 64]. Academic research in peer-to-peer (P2P) systems has concentrated largely on efficiency [65], scalability [66], robustness [67], and security [68] on P2P architecture, services such as indexing and search [69], dissemination [70] for applications running on top of P2P systems, or even many of the above [71].

As a case study to test the feasibility of the architectural decomposition and formal techniques involved in **Paper I** and **Paper II**, *Gnutella*, was chosen as the appropriate candidate as the starting point. M. Ripeanu [72] studied the P2P architecture via Gnutella also, and it was the first kind of systematic description of Gnutella-like systems in the literature. I. Ivkovic [73] further analyzed the Gnutella protocol, and advised several research directions and proposals.

In the above two works, the decentralized content distribution architecture and the *de facto* Gnutella protocol were depicted via a reverse-engineering approach. It should be admitted that they greatly influenced our understanding of P2P, and it was also the foundation on which the architecture framework in our papers was based. In our works, we took one step further, from these ad hoc descriptions of the system to a more systematic requirement document. From a software engineering practitioner's view, we refined the requirement presented in the above two papers and further formalized it into a formal specification which can be followed by other software engineers. As a testimony, a Gnutella client was developed with our specification [9].

Generally, the proposed P2P architecture in this thesis was inspired [74]. D. Barkai argued that the common services in P2P systems can be thought of as a middleware layer. One of the principal advantages of a common middleware is that application developers will no longer be required to keep creating the same basic services over and over again. Interoperability means that P2P applications can communicate across different software environments. Peers running Windows or Linux or any other operating system can share P2P applications. Applications using different programming languages can communicate and be integrated via a common middleware. And common middleware provides a mechanism and an infrastructure for incorporating devices other than PCs and servers. Such devices include wireless and handheld products and various network appliances.

Though Intel didn't release any API with the above ideas, we borrowed this middleware idea into our specification. The multi-tiers in that paper imply a modular design and component-based architecture. In our papers, though we didn't mention the middleware layer explicitly, the main components that form the intermediate layer, which glue the network layer and the application layer together, can be deemed a specific case of middleware. Our work can be deemed as a refined specification of the architecture proposed from the above paper. Naturally, one future work for us would be the development of a middleware providing common services in P2P systems.

Besides Gnutella, several other typical P2P applications were also extensively studied. K. W. Ross *et al.* studied *Kazaa*, one of the important applications in the Internet today, both in terms of number of participating users and in traffic volume.

They built a measurement platform for collecting and measuring Kazaa's signaling traffic. These measurements provide insight on Kazaa's architecture, protocol, and overlay behavior [75]. R. Steinmetz *et al*. studied *eDonkey2000* file-sharing network [89], which was one of the most successful peer-to-peer file-sharing applications in Germany. They described the eDonkey protocol and measurement results on network/transport layer and application layer that were made with the client software and with an open-source eDonkey server extended for these measurements [76].

We believe the above two works are valuable, partly because of the popularity of *Kazaa* and *eDonkey2000*. For instance, according to a survey from the computing center of Turku University [151], the above two applications consume approximate 89% bandwidth of the whole local network traffic within one month observation period. As a drawback of our papers, we didn't present a quantitative analysis of the network traffic for Gnutella [152]. As an integrated part to verify our proposed architecture, some measurement studies would be done as a future work for the implemented prototype [9].

Another significant P2P network is *JXTA*, also known as an open-source P2P platform [90]. JXTA technology is a network programming and computing platform that is designed to solve a number of problems in modern distributed computing, especially in the area broadly referred to as peer-to-peer computing, or peer-to-peer networking, or simply P2P [77].

Though the impact of JXTA has not become as great as expected yet, we believe it is a promising candidate for the future P2P world. Some future work could be done on studying the software architecture of JXTA and its implications on the very large scale distributed systems, or so-called global computing [150].

The ubiquity of the Internet has made possible a universal storage space that is distributed among the participating end computers (peers) across the Internet. All peers assume equal role and there is no centralized server in the space. Such architecture is collectively referred to as the peer-to-peer (P2P) storage architecture. Several P2P storage systems have been built; examples include Freenet [78], OceanStore [79], PAST [80] and CFS [81]. Applications such as groupware, e-mail, storage repositories for scientific data and visualization, searching network, distributed file systems and wide-scale shared states can benefit from such storage systems [82, 83, 84].

The massive P2P storage system in **Paper III** was generally inspired by *OceanStore*. It is UC Berkeley's vision on providing global-scale persistent data [91]. OceanStore is a global persistent data store designed to scale to billions of users. It provides a consistent, highly-available, and durable storage utility atop an infrastructure comprised of untrusted servers. Any computer can join the infrastructure, contributing storage or providing local user access in exchange for economic compensation. Users need only subscribe to a single OceanStore service provider, although they may consume storage and bandwidth from many different providers. The providers automatically buy and sell capacity and coverage among themselves, transparently to the users. The utility model thus combines the resources from federated systems to provide a quality of service higher than that achievable by any single company.

With the same idea in *OceanStore* that the future trend of large scale storage systems would Internet-based, we, however, implemented a prototype that is more suitable for the home or small office scale storage. In general, the prototype is a hybrid P2P system, which has some centralized index nodes and many decentralized data nodes. As a main drawback, the hybridity in our design exposes us to the potential risk that any compromise in the index nodes would degrade the availability of the whole system. While in OceanStore, several advanced techniques, DHT, replication, and fault-tolerant protocol, are used to assure the high dependability of the storage system. As a future work, more research would be done, focusing on the dependability issues of the proposed system.

Pastry [85] is Microsoft Research's P2P substrate [86] for massive storage. From this foundation, they built *PAST* [87], a large-scale, peer-to-peer archival storage utility that provides scalability, availability, security and cooperative resource sharing. Files in PAST are immutable and can be shared at the discretion of their owner.

The Cooperative File System (*CFS*) [81] is MIT's peer-to-peer readonly storage system that provides provable guarantees for the efficiency, robustness, and load-balance of file storage and retrieval. It employs a P2P hash-based system Chord [88], and implements CFS layers storage on top of an efficient distributed hash lookup algorithm.

Those papers suggest the possibility of building a file system with P2P on top of the Internet. The further implication is a network-centric operation system based on the proposed file system, instead of a PC-centric operation system nowadays. Therefore, another future work for us would be infrastructure: since in scientific research, the data storage demand has reached the PB level [125, 126], it is time to rethink the atomic building blocks in a large scale distributed storage system. It would be promising to construct an intelligent network storage system based on *object*, instead of disk or cluster.

Ad hoc networks [92] are a key factor in the evolution of wireless communications. Self-organized ad hoc networks of PDAs or laptops are used in disaster relief, conference, and battlefield environments. These networks inherit the traditional problems of wireless and mobile communications, such as bandwidth optimization, power control, and transmission-quality enhancement. In addition, their multihop nature and the possible lack of a fixed infrastructure introduce new research problems such as network configuration, device discovery, and topology maintenance, as well as ad hoc addressing and self-routing [93, 94, 95, 96, 97]

In the literature several related work to **Paper IV** can be found [98, 99, 100, 101]. For instance, ETH prototyped a testbed [99] that is suitable for the deployment of large populations of heterogeneous distributed mobile communication and information systems to support a "smart" networked environment in everyday objects via Bluetooth-based mobile ad hoc networks. Besides *Bluetooth* and *WiFi* (802.11x), we argue that another emerging new IEEE standard *ZigBee* (802.15.4) would also be promising for the ubiquitous networking [102].

The above papers mostly address the algorithmic aspects of ad hoc networking, since the routing and connectivity issues in MANET have been major challenges for a decade. While in our paper, we are mostly interested in the software architecture aspect of MANET applications. We took a component based framework in that paper and further specified each component with formal methods. As a drawback, there is no specific algorithm presented, neither further quantitative measurement study, which would be a future work. With the prevalence of wireless sensor networks, which is MANET in nature, we plan to extend our framework to wireless sensor networks, and some application scenarios are being investigated. In that case, dependability becomes a must, partly because data from distributed sensors can be highly sensitive and safety-critical.

The synergy between P2P overlay and MANET overlay in **Paper V** and **Paper VI** was generally inspired [103]. Y. C. Hu *et al.* studied the possibility of the merge of the P2P technology and the MANET technology, and proposed Dynamic P2P Source Routing (DPSR), a new routing protocol for MANET that exploits the synergy between P2P and MANET for increased scalability. The first trial system of this kind, as known, is *Proem* [104], which is a P2P platform for developing mobile P2P applications, but it seems to be a rough one and only IEEE 802.11b in ad hoc mode is supported. *7DS* [105] is an attempt to enable P2P resource sharing and information dissemination in mobile environments, and the proposed architecture is cited by many later works. *ORION* [106] is a special-purpose approach for P2P file sharing tailored to MANET. ORION combines application-layer query processing with the network layer process of route discovery, substantially reducing control overhead and increasing search accuracy.

Compared to the above papers, our papers are more focused on the network architecture, partly because we believe the architecture is one of the most critical issues in any complex system. Though the proposed architecture seems promising, it is still in its early stage. One important issue lacked in our framework is the management of trust and privacy, which is a critical and challenging topic in ubiquitous computing. Taking a step further, in a derived middleware from the specification, it means two important open problems: timely interaction and access control [155]. As work in progress [156], we have carried out the research on formalizing trust in a fully decentralized distributed computing environment. In the future, more work would be done on realizing trust domains, scalable from sensor networks to grid applications, in which decisions are based on evidence, mitigated by trust and privacy requirements.

For the cross-layer routing of P2P and MANET, B. Bhargava *et al.* conducted a preliminary survey of cross-layer schemes [107]. Virtual Paths Routing (*VPR*) Protocol [108] was proposed for Ad Hoc wireless networks to provide correct, efficient, and highly dynamic route creation and maintenance between mobile nodes. Another cross-layer protocol [109] based on Pastry, emphasized the cross-layer interaction with a pro-active routing protocol at the network layer.

Though we all agree that cross-layer routing will have potential advantages, there is actually *no* off-shelf cross-layer protocol at the moment, as far as known. In our papers, research in this area has just started, and future work would be done on further investigating the possibility to introduce DHT to cross-layer routing schemes.

The general performance issues on P2P and MANET were discussed [110, 111, 112, 113, 114]. In the literature, [115, 116, 117, 118, 119] extensively discussed the measurement, modeling, and analysis of the peer-to-peer file-sharing with different approaches. Recently, B. Bakos *et al*. analyzed a Gnutella-style protocol query engine on mobile networks with different topologies [120], and T. Hossfeld *et al*. conducted a simulative performance evaluation of mobile P2P file-sharing [121].

It should be admitted that our model is mostly based on [119], where R. Srikant *et al*. proposed a dynamic fluid approach in modeling P2P traffic. The classification of peer roles in that paper influenced our thinking of mobile P2P system, because in a mobile environment, this classification would have an even greater impact on the system behavior. Since this dynamic fluid approach is also suitable for mobile settings, we applied that model to a mobile P2P system, where old notions would get new practical meanings. We pointed out some key concepts such as *effective bandwidth*, *bottleneck bandwidth*, which was implied in that paper, but not explicitly stated. Though it works quite fine according to the experiment results, there are two important issues lacked in our model, which would be the future work: (1) the effect of network topology on performance evaluation (2) the effect of heterogeneous bandwidth capacity of nodes on the system performance.

The stepwise system development by refinement in Figure 2, proposed by R.J. Back and K. Sere via Action Systems [4, 17], is also employed by G. C. Roman via UNITY [122]. With the prevalence of mobile computing, Mobile UNITY [123] was proposed as a new model for specifying and reasoning about concurrent systems that contain dynamically reconfiguring components. Such reasoning is carried out axiomatically, in the style of standard UNITY. Recently, *Context UNITY* [124] was proposed as a further step from Mobile UNITY to address the problems in the context-aware computing, which is a related work to **Paper VII**.

Compared to Context UNITY, it should be admitted that one main drawback in our formalism is the ad hoc treatment of time, partly because the foundation of our framework, refinement calculus [18], is not a temporal logic. Therefore in our paper, time is introduced as an external variable, which makes it hard to model some real-time requirements and dynamic properties of the system. M. Ronkko proposed an extension of action systems to address the discrete-time issues for modeling hybrid systems [157]. As a future work, we would investigate the possibility of applying this technique to our framework, to achieve a better treatment of time.

Many researchers agree that nowadays the key challenge in ubiquitous computing is context-awareness [158]. Some discussions on context-aware computing can be found [16, 28, 49, 127, 128]. Since we have gained experiences in the previous ubiquitous networking research, we envisage a general direction for us in the future aiming at building a context-aware middleware with the architectures and communication models from P2P and MANET [153, 154].

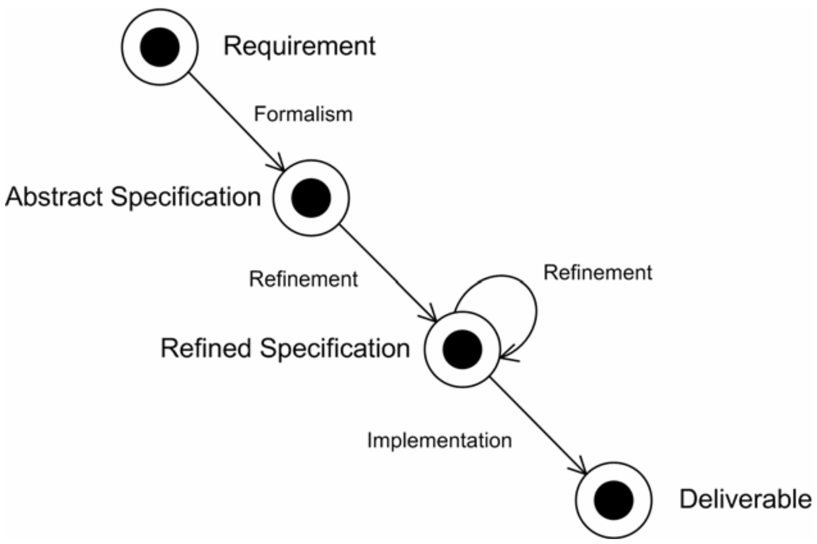


Figure 2. Stepwise System Development by Refinement

If we date back to the computing history [57], Java was created as a part of the Green project [129] specifically for an embedded device, a handheld wireless PDA. The device was never released as a product, but Java was launched as the new language for the Internet. Over the time, Java became very popular to build desktop applications, web services and ubiquitous systems. On the other hand, Java technology is compromised, and the major drawback is the lack of acceptable runtime performance.

In order to improve Java bytecode execution by hardware, Java coprocessor, as in **Paper VIII** was proposed to work in conjunction with a main processor. In the literature, Hard-Int [130] was proposed as an additional architecture for a standard RISC processor to speed up a JVM interpreter, and simple Java bytecodes are translated to a sequence of RISC instructions. Delft-Java [131] is a processor for multimedia applications in Java, i.e. a RISC processor extended with DSP capabilities and Java specific instructions. Simple JVM instructions are dynamically translated to the DELFT instruction set. Jazelle [132] is an extension of the ARM 32-bit RISC processor, which is integrated into the same chip as the ARM processor. Java bytecodes are translated into sequences of ARM instructions. Another significant related work is a co-designed JVM based on FPGA technology, emphasizing reconfigurability [133].

Compared to the above works, our design goal is more biased toward energy consumption than speed, because of the nature of ubiquitous devices. We tried to use some asynchronous circuit design techniques to achieve this goal, but this work is still in progress. At moment, the coprocessor is validated in the SystemC [159] model. This high-level simulation model is more a proof of concept, so no overall quantitative speedup rate and energy consumption estimation are available at this stage. A hardware implementation of this prototype in FPGA is the foremost task in our future research.

Formal hardware verification has recently attracted considerable interests due to the prevalence of ubiquitous computing. The need for correct designs for safety-critical applications, coupled with the major cost associated with products delivered late, are the two main factors behind this [134]. In the literature, several general surveys on hardware verification can be found [135, 136, 137, 138, 139]. In the past, hardware verification methods have divided into two well-established approaches: model checking [140] and theorem proving [141].

Provided that a design can be specified as a finite state machine, model checking can exhaustively test whether the state machine satisfies a given correctness property by searching the graph of all possible states that can be reached from a set of initial states. On the contrary, theorem proving does not search the state space of a specification directly, but instead searches through the space of correctness proofs that a specification satisfies a given correctness property.

In theory, theorem proving is more attractive because it can deal directly with infinite state space [142]. Therefore we adopted this approach in **Paper IX**. Though the case study is so trivial that it is also easy to be completed via model checking [160], realistic designs of commercial ubiquitous devices often contain thousands or millions of state variables, far exceeding the reach of current model checking algorithms. Theorem proving does not directly depend on the size of a specification's state, so in principle specifications of unbounded size can be verified. However, in practice the size of the proof space is too large to be searched automatically.

As an extra finding of that case study, we realize that model checking and theorem proving are complementary technologies and must be integrated to successfully tackle realistic system designs. In fact, the practical formal verification in hardware design is usually approaches that combine theorem proving and model checking [143]. For instance, during the past years, Intel has verified many major processors it produced [144], including the famous floating point error of Pentium series [145] and the new released XScale series [146] targeted for ubiquitous devices. IBM also has a long tradition of hardware verification [147] and applied various hardware verification methods to its cutting-edge Blue Gene series [148].

5 Final Words

Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by many people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of *calm technology*, when technology recedes into the background of our lives. Alan Kay[3] calls this *Third Paradigm* computing. This has required new work in operating systems, user interfaces, networks, wireless, displays, and many other areas. We call it "ubiquitous computing". This is different from PDA's, notebooks, or information at your fingertips. It is invisible, everywhere computing that does not live on a personal device of any sort, but is in the woodwork everywhere[4].

In the coming years, the technology industry will be working hard to enable the devices, software and services in your life to work together in a more seamless way, creating new ways for technology to enhance your life. Although the changes you will see will be gradual, the differences between the computing experiences of today and how you'll use them five years from now will be like night and day[5].

[3] "The best way to predict the future is to invent it." (1971).
[4] Mark Weiser's speech at Xerox PARC (1988).
[5] Bill Gate's speech at CeBIT (2005).

Bibliography

[1] M. Weiser. The Computer for the Twenty-First Century. In *Scientific American*, Sept. 1991.
[2] M. Weiser. Some Computer Science Problems in Ubiquitous Computing. In *Communications of the ACM*, July 1993.
[3] M. Weiser. Hot Topics: Ubiquitous Computing. In *IEEE Computer*, Oct. 1993.
[4] K. Sere. *Stepwise derivation of parallel algorithms.* Academic dissertation, Abo Akademi, 1990.
[5] J. A. Hall. Seven myths of formal methods. In *IEEE Software* 7(5), pp. 11-19, Sept. 1990.
[6] J. P. Bowen, M. G. Hinchey. Seven more myths of formal methods. In *IEEE Software* 12(4), pp. 34-41, July 1995.
[7] S. Nygard. *Implementation of a transaction-based Gnutella search engine.* Master thesis, Abo Akademi, 2004.
[8] Gnutella: http://www.gnutella.com/
[9] N. Iqbal. *Development of Gnutella Client with Formal Specification*. Master thesis, Abo Akademi, 2003.
[10] Napster: http://www.napster.com/
[11] Kazaa: http://www.kazaa.com/
[12] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, J. Kubiatowicz. Pond: the OceanStore Prototype. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, March 2003.
[13] J. R. Abrial. *The B-Book: Assigning Programs to Meaning*. Cambridge University Press, 1996.
[14] J. Ni. *Towards a Systematic Design of Applications for Ad Hoc networks*. Master thesis, Abo Akademi, 2003.
[15] Bluetooth SIG. *Bluetooth Specification.* http://www.bluetooth.com/
[16] A.K. Dey, G.D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness*, April 2000.
[17] R.J. Back, K. Sere. From Action Systems to Modular Systems. In *Software - Concepts and Tools*, (1996) 17: 26-39.
[18] R.J. Back, J. Wright. *Refinement Calculus: A Systematic Introduction.* Graduate Texts in Computer Science, Springer-Verlag, 1998.
[19] A.Gupta. Formal Hardware Verification Methods: A Survey. In *Journal of Formal Methods in System Design*, vol. 1, pp. 151 - 238, 1992
[20] M.Gordon. *HOL: A Machine Oriented Formulation of Higher Order Logic*. Technical Report 68, Computer Laboratory, University of Cambridge, 1985
[21] N.Shankar, S.Owre, J.M.Rushby, D.W.J.Stringer-Calvert. *PVS System Guide*: http://pvs.csl.sri.com/doc/pvs-system-guide.pdf
[22] M. Weiser. The world is not a desktop. In *ACM Interactions*, Jan. 1994.
[23] P. Dourish. *Where The Action Is: The Foundations of Embodied Interaction*. MIT Press, 2001.

[24] V. Bellotti, P. Dourish, A. MacLean. *From Users' Themes to Designers DReams: Developing a Design Space for Shared Interactive Technologies*. Technical Report EPC-91-112, Rank Xerox EuroPARC, Cambridge, UK, 1991.
[25] J. Meyer, L. Staples, S. Minneman, M. Naimark, A. Glassner. Artists and technologists working together. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*, San Francisco, California, 1998.
[26] R. Want, B. Schilit, N. Adams, R. Gold, D. Goldberg, K. Petersen, J. Ellis, M. Weiser. An Overview of the Parctab Ubiquitous Computing Experiment. In *IEEE Personal Communications*, Vol. 2, No.6, pp. 28-43, Dec. 1995.
[27] B. Opshaug. *Investigating attitudes towards ubiquitous information systems and the possibilities to overcome skepticism towards these systems through corporate branding*. Technical Report, PD9, Institutt for Produktdesign, NTNU, Norway, 2002.
[28] T. Strang, C. Linnhoff-Popien. A Context Modeling Survey. In *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning And Management*, Nottingham, England, Sept. 2004.
[29] H. Ailisto, A. Kotila, E. Strömmer. *Ubicom applications and technologies*. VTT Research Notes 2201, VTT, Finland, 2003.
[30] G. D. Abowd, E. D. Mynatt. Charting past, present, and future research in ubiquitous computing. In *ACM Transactions on Computer-Human Interaction*, Volume 7, Issue 1, March 2000.
[31] U. Saif. *Architectures for ubiquitous systems*. Technical Report, UCAM-CL-TR-527, Computer Laboratory, University of Cambridge.
[32] C. Endres, A. Butz, A. MacWilliams. A Survey of Software Infrastructures and Frameworks for Ubiquitous Computing. In *Mobile Information Systems Journal*, January-March, 2005.
[33] MIT Project Oxygen: http://oxygen.lcs.mit.edu/
[34] A. Champaneria. *PADCAM: A Portable Human-Centric System for Handwriting Capture*. Master Thesis, MIT, 2002.
[35] A. C. Kao. *Design and Implementation of a Generalized Device Interconnect*. Master Thesis, MIT, 2002.
[36] A. Narayanswamy. *Real-time Visualization of Abstract Relationships Between Devices*. Master Thesis, MIT, 2003.
[37] J. Brunsman. *The Application and Design of the Communication Oriented Routing Network*. Master Thesis, MIT, 2003.
[38] G. Eguchi. *Extending CORE for Real World Appliances*. Master Thesis, MIT, 2003.
[39] HP Cooltown: http://www.cooltown.com/
[40] J. Barton, T. Kindberg. *The Cooltown User Experience*. Technical Report HPL-2001-22, HP Labs, 2001.
[41] P. Debaty, P. Goddi, A. Vorbau. *Integrating the Physical World with the Web to Enable Context-Enhanced Services*. Technical Report HPL-2003-192, HP Labs, 2003.

[42] S. Pradhan, C. Brignone, J. H. Cui, A. McReynolds, M. Smith. *Websign: Hyperlinks from a Physical Location to the Web*. Technical Report HPL-2001-140, HP Labs, 2001.
[43] R. Hull, J. Reid, A. Kidd. *Experience Design in Ubiquitous Computing*. Technical Report HPL-2002-115, HP Labs, 2002.
[44] J. J. Barton, V. Vijayaraghavan. *UBIWISE, A Simulator for Ubiquitous Computing Systems Design*. Technical Report HPL-2003-93, HP Labs, 2003.
[45] CMU Project Aura: http://www-2.cs.cmu.edu/~aura/
[46] D. Garlan, D. Siewiorek, A. Smailagic, P. Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. In *IEEE Pervasive Computing*, April-June 2002.
[47] J. P. Sousa, D. Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. In *Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, Aug. 2002.
[48] S. W. Cheng, D. Garlan, B. Schmerl, J. P. Sousa, B. Spitznagel, P. Steenkiste, N. Hu. Software Architecture-based Adaptation for Pervasive Systems. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS'02)*, April, 2002.
[49] G. Judd, P. Steenkiste. Providing Contextual Information to Pervasive Computing Applications. In *Proceedings of IEEE International Conference on Pervasive Computing (PERCOM)*, Dallas, March, 2003.
[50] U. Hengartner, P. Steenkiste. Implementing access control to people location information. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies (SACMAT'04)*, Yorktown Heights, June 2004.
[51] EU Project Disappearing Computer: http://www.disappearing-computer.net/
[52] T. Rodden, A. Crabtree, T. Hemmings, B. Koleva, J. Humble, K. P. Åkesson, P. Hansson. Between the dazzle of a new building and its eventual corpse: assembling the ubiquitous home. In *Proceedings of the 2004 ACM Symposium on Designing Interactive Systems*, Cambridge, Massachusetts, Aug. 2004.
[53] H. Hutchinson, H. Hansen, N. Roussel, B. Eiderbäck, W. Mackay, B. Westerlund, B. B. Bederson, A. Druin, C. Plaisant, M. Beaudouin-Lafon, S. Conversy, H. Evans. Technology probes: inspiring design for and with families. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2003.
[54] H. W. Gellersen, A. Schmidt, M. Beigl. Multi-sensor context-awareness in mobile devices and smart artifacts. In *ACM Mobile Networks and Applications*, Volume 7, Issue 5, Oct. 2002.
[55] D. Stanton, T. Pridmore, V. Bayon, H. Neale, A. Ghali, S. Benford, S. Cobb, R. Ingram, C. O'Malley, J. Wilson. Classroom collaboration in the design of tangible interfaces for storytelling. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2001.
[56] M. Büscher, M. A. Eriksen, J. F. Kristensen, P. H. Mogensen. Ways of grounding imagination. In *Proceedings of Participatory Design Conference (PDC) 2004*, Toronto, Canada, July 2004.
[57] B. Carlson, A. Burgess, C. Miller, L. Bauer. Timeline of Computing History. In *IEEE Computer*: http://www.computer.org/computer/timeline/

[58] SETI@home: http://setiathome.ssl.berkeley.edu/
[59] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly Press, 2001.
[60] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, Z. Xu. *Peer-to-Peer Computing*. Technical Report HPL-2002-57R1, HP Labs, 2002.
[61] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. In *IEEE Communications*, March 2004.
[62] A. T. Stephanos, S. Diomidis. A survey of peer-to-peer content distribution technologies. In *ACM Computing Surveys*, Volume 36, Issue 4, Dec. 2004.
[63] D. Wallach. A Survey of Peer-to-Peer Security Issues. In *International Symposium on Software Security*, 2002.
[64] B. Yang, H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. In Proceedings of the 27th International Conference on Very Large Data Bases, 2001.
[65] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM*, 2001.
[66] P. Felber, K.W. Ross, E.W. Biersack, L. Garces-Erice, G. Urvoy-Keller. Structured Peer-to-Peer Networks: Faster, Closer, Smarter. In *IEEE Data Engineering Bulletin*, 28(1):55-62, 2005.
[67] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *ACM SIGCOMM*, 2003.
[68] M. Srivatsa, L. Liu. Vulnerabilities and Security Threats in Structured Overlay Networks: A Quantitative Analysis. In *Proceedings of the 20th IEEE Annual Computer Security Applications Conference*, 2004.
[69] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. R. Karger, R. Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. In *Proceedings of IPTPS*, 2003.
[70] M. Koubarakis, C. Tryfonopoulos, S. Idreos, Y. Drougas. Selective information dissemination in P2P networks: problems and solutions. In *ACM SIGMOD*, Volume 32, Issue 3, 2003.
[71] M. Roussopoulos, M. Baker, D. S. H. Rosenthal, T. J. Giuli, P. Maniatis, J. Mogul. 2 P2P or Not 2 P2P? In *Proceedings of the Third International Workshop on Peer-to-Peer Systems*, 2004.
[72] M. Ripeanu. *Peer-to-peer architecture case study: Gnutella network*. Technical Report TR-2001-26, University of Chicago, Department of Computer Science, 2001.
[73] I. Ivkovic. *Improving Gnutella Protocol: Protocol Analysis and Research Proposals*. Technical report, LimeWire LLC, 2001.
[74] D. Barkai. An Introduction to Peer-to-Peer Computing. In *Intel Developer Update*, Intel Labs, Feb. 2000.
[75] J. Liang, R. Kumar, K. W. Ross. *Understanding KaZaA*. Technical Report, Polytechnic University, New York, 2004.

[76] O. Heckmann, A. Bock, A. Mauthe, R. Steinmetz. The eDonkey File-Sharing Network. In *Workshop on Algorithms and Protocols for Efficient Peer-to-Peer Applications*, Informatik 2004, Sept. 2004.
[77] L. Gong. Project JXTA: A Technology Overview. In *IEEE Internet Computing*, May/June, 2001.
[78] I. Clarke, O. Sandberg, B. Wiley, T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Proceedings of Workshop on Design Issues in Anonymity and Unobservability*, July 2000.
[79] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, Nov. 2000.
[80] A. Rowstron, P. Druschel. Storage Management and Caching in PAST, A Large-Scale, Persistent Peer-to-Peer Storage Utility. In *ACM Symposium on Operating Systems Principles*, Oct. 2001.
[81] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica. Wide-Area Cooperative Storage with CFS. In *ACM Symposium on Operating Systems Principles*, Oct. 2001.
[82] W. J. Bolosky, J. R. Douceur, D. Ely, M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, June 2000.
[83] H. C. Hsiao, C. T. King. Modeling and Evaluating Peer-to-Peer Storage Architectures. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, April 2002.
[84] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, R. Campbell. A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems. In *Proceedings of IEEE International Conference on Information Technology (ITCC)*, April 2005.
[85] A. Rowstron, P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001.
[86] M. Castro, M. Costa, A. Rowstron. *Peer-to-peer overlays: structured, unstructured, or both?* Technical Report MSR-TR-2004-73, Microsoft Research, 2004.
[87] P. Druschel, A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, May 2001.
[88] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. In *IEEE/ACM Transactions on networking*, Vol. 11, No. 1, Feb. 2003.
[89] eDonkey2000: http://www.edonkey2000.com/
[90] JXTA: http://www.jxta.org/
[91] OceanStore: http://oceanstore.cs.berkeley.edu/
[92] F. Baker. *An outsider's view of MANET.* Internet Engineering Task Force (IETF) document, 17 March 2002.

[93] S. Basagni *et al*. (eds.). *Mobile Ad Hoc Networking*, IEEE Press, 2003.
[94] M. Frodigh *et al*. Wireless Ad Hoc Networking: The Art of Networking without a Network. In *Ericsson Review*, No. 4, 2000.
[95] Z. J. Haas, *et al.* (eds.). *Special Issue on Wireless Ad Hoc Networks*, IEEE Journal on Selected Areas in Communications, Vol. 17, No. 8, 1999.
[96] C. E. Perkins. *Ad Hoc Networking*. Addison-Wesley, 2001.
[97] S. Sesay, Z. Yang, J. He. A Survey on Mobile Ad Hoc Wireless Network. In *Information Technology Journal*, 3 (2): 168-175, 2004
[98] F. Kargl, S. Ribhegge, S. Schlott, M. Weber. Bluetooth-based Ad-Hoc Networks for Voice Transmission. In *HICSS'03*, 2003.
[99] J. Beutel, O. Kasten, M. Ringwald, F. Siegemund, L. Thiele. *Bluetooth Smart Nodes for Ad-hoc Networks*. TIK Report Nr. 167, ETH, 2003.
[100] E. Ferro, F. Potorti. Bluetooth and Wi-Fi wireless protocols: A survey and a comparison. In *IEEE Wireless Communications*, vol. 12, no. 1, pp. 12-26, Feb. 2005.
[101] B. Zhen, J. Park, Y. Kim. *Scatternet Formation of Bluetooth Ad Hoc Networks*. Technical Report, Samsung Labs, 2003.
[102] J. Zheng, M. J. Lee. Will IEEE 802.15.4 make ubiquitous networking a reality?: A discussion on a potential low power, low bit rate standard. In *IEEE communications*, vol. 42, no. 6, pp. 140-146, June 2004.
[103] Y. C. Hu, S. M. Das, H. Pucha. Exploiting the Synergy between Peer-to-Peer and Mobile Ad Hoc Networks. In *Proceedings of HotOS-IX: Ninth Workshop on Hot Topics in Operating Systems*, Hawaii, May, 2003.
[104] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, Z. Segall. When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks. In *Proc. 1st International Conference on Peer-to-Peer Computing (P2P 2001)*, Linkoping, Sweden, Aug. 2001.
[105] M. Papadopouli, H. Schulzrinne. A Performance Analysis of 7DS a Peer-to-Peer Data Dissemination and Prefetching Tool for Mobile Users. In *Advances in wired and wireless communications, IEEE Sarnoff Symposium Digest*, 2001.
[106] A. Klemm, Ch. Lindemann, O. Waldhorst. A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks. In *Proc. IEEE Vehicular Technology Conf.*, Orlando, FL, Oct. 2003.
[107] G. Ding, B. Bhargava. Peer-to-peer File-sharing over Mobile Ad hoc Networks. In *Proc. the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, Florida, 2004.
[108] A. H. Altalhi, G. G. Richard. Virtual Paths Routing: A Highly Dynamic Routing Protocol for Ad Hoc Wireless Networks. In *Proc. the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, Florida, 2004.
[109] M. Conti, E. Gregori, G. Turi. Towards Scalable P2P Computing for Mobile Ad Hoc Networks. In *Proc. the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, Florida, 2004.
[110] H. Birck, O. Heckmann, A. Mauthe, R. Steinmetz. The Two-Step Overlay Network Simulation Approach. In *Proceedings of SoftCOM*, Split, Croatia, Oct. 2004.

[111] L. Voinea, A. Telea, J. J. Wijk. EZEL: a Visual Tool for Performance Assessment of Peer-to-Peer File-Sharing Network. In *Proc. IEEE Symposium on Information Visualization (INFOVIS '04)*, Texas, 2004.
[112] P. Triantafillou, C. Xiruhaki, M. Koubarakis, N. Ntarmos. Towards High-Performance Peer-to-Peer Content and Resource Sharing Systems. In *First Biennial Conference on Innovative Data Systems Research (CIDR 2003),* California, Jan. 2003.
[113] J. Hsu, S. Bhatia, M. Takai, R. Bagrodia, M. Acriche. Performance of Mobile Ad Hoc Networking Routing Protocols in Realistic Scenarios. In *Proceedings of MILCOM '03,* Boston, Oct. 2003.
[114] E. M. Royer, C. K. Toh. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks, In *IEEE Personal Communications*, April 1999.
[115] Z. Ge, D. Figueiredo, S. Jaiswal, J. F. Kurose, D. Towsley. Modeling peer-to-peer file sharing systems. In *Proc. IEEE Infocom*, 2003.
[116] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. 19th ACM symposium on Operating systems principles,* 2003.
[117] F. Clevenot, P. Nain. A Simple Fluid Model for the Analysis of the Squirrel Peer-to-Peer Caching System. In *Proc. IEEE Infocom*, 2004.
[118] X. Yang, G. Veciana. Service Capacity of Peer to Peer Networks. In *Proc. IEEE Infocom*, 2004.
[119] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In *Proc. ACM SIGCOMM*, 2004.
[120] B. Bakos, G. Csucs, L. Farkas, J. K. Nurminen. Peer-to-peer protocol evaluation in topologies resembling wireless networks. An Experiment with Gnutella Query Engine. In *Proc. International Conference on Networks,* Sydney, Oct., 2003.
[121] T. Hossfeld, K. Tutschku, F. U. Andersen, H. Meer, J. Oberender. *Simulative Performance Evaluation of a Mobile Peer-to-Peer File-Sharing System*. Research Report 345, University of Wurzburg, Nov. 2004.
[122] H. C. Cunningham, G. C. Roman.A UNITY-Style Programming Logic for a Shared Dataspace Language. In *IEEE Transactions on Parallel and Distributed Systems* 1, No. 3, July 1990.
[123] G. C. Roman, P. J. McCann, J. Y. Plun. Mobile UNITY: Reasoning and Specification in Mobile Computing. *In ACM Transactions on Software Engineering and Methodology* 6, No. 3, July 1997.
[124] G. C. Roman, C. Julien, J. Payton. A Formal Treatment of Context-Awareness. In *Proceedings of the 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004),* Lecture Notes in Computer Science 2984, Springer, March/April 2004.
[125] Fermilab: http://www.fnal.gov/
[126] CERN: http://www.cern.ch/
[127] G. Chen, D. Kotz. *A survey of context-aware mobile computing research*. Dartmouth College Technical Report TR2000-381, Nov. 2000.
[128] K. Mitchell. *Supporting the Development of Mobile Context-Aware Computing*. Academic Dissertation. Lancaster University, 2002.

[129] J. Gosling. A Brief History of the Green Project:
http://today.java.net/jag/old/green/
[130] R. Radhakrishnan. *Microarchitectural Techniques to Enable Efficient Java Execution*. Academic Dissertation, University of Texas at Austin, 2000.
[131] C. J. Glossner. *The DEFLT-JAVA Engine*. Academic Dissertation, Delft University of Technology, 2001.
[132] Jazelle – ARM Architecture Extensions for Java Applications. White Paper, ARM.
[133] K. B. Kent. *The Co-Design of VirtualMachines Using Reconfigurable Hardware*. Academic Dissertation, University of Victoria, 2003.
[134] C. J. H. Seger. An *Introduction to Formal Verification*. Technical Report 92-13, UBC, Canada, 1992.
[135] T. Kropf. *Introduction to Formal Hardware Verification*. Springer-Verlag, 2000.
[136] T. Kropf. *Formal Hardware Verification : Methods and Systems in Comparison*. Lecture Notes in Computer Science 1287, Springer-Verlag, 1997.
[137] Survey of Formal Verification. In *IEEE Spectrum*, June 1996.
[138] C. Kern, M. Greenstreet. Formal Verification in Hardware Design: A Survey. In *ACM Transactions on Design Automation of Electronic Systems*, Vol. 4, April 1999.
[139] M. Yoeli. *Formal Verification of Hardware Design*, IEEE Computer Society Press, 1991.
[140] E. M. Clarke, O. Grumberg, D. Peled. *Model Checking*, MIT Press, 2000.
[141] M. Kaufmann, P. Manolios, J. S. Moore. *Computer-Aided Reasoning: An Approach*, Kluwer Academic Publishers, 2000.
[142] E.Clarke, J.Wing. *Formal Methods: State of the Art and Future Directions*. CMU Computer Science Technical Report, CMU-CS-96-178, 1996
[143] R. B. Jones, J. W. O´Leary, C. J. H. Seger, M. D. Aagaard, T. F. Melham. Practical formal verification in microprocessor design. In *IEEE Design and Test*, 18(4): 16-25, July/August 2001.
[144] J. Harrison. Formal Verification at Intel. In *IEEE LICS 2003*, Canada, June 2003.
[145] J. O´Leary, X. Zhao, R. Gerth, C. J. H. Seger. Formally verifying IEEE compliance of floating point hardware. In *Intel Technology Journal*, 1999.
[146] S. K. Srinivasan, M. N. Velev. Formal Verification of an Intel XScale Processor Model with Scoreboarding, Specialized Execution Pipelines, and Imprecise Data-Memory Exceptions. In *Formal Methods and Models for Codesign (MEMOCODE '03)*, June 2003.
[147] S. Ben-David, C. Eisner, D. Geist, Y. Wolfsthal. Model Checking at IBM. In *Journal of Formal Methods in System Design*, 22 (2), 2003.
[148] *Special Issue on Blue Gene*. IBM Journal of Research and Development, Vol. 49, No. 2/3, 2005.
[149] D. Garlan. Software Architecture: a Roadmap. In *The Future of Software Engineering*, A. Finkekstein (Ed), ACM Press, 2000.

[150] J. Bacon. Expectations and reality in distributed systems. In *Proceedings of IASTED International Conference on Parallel and Distributed Computing and Networks*, Insbruck Austria, February 2005.
[151] University of Turku. Data communication statistics:
http://www.cc.utu.fi/english/network/stats/index.html
[152] D. Zeinalipour-Yazti, T. Folias. *Quantitative Analysis of the Gnutella Network Traffic*. Technical Report TR-CS-89, Dept. of Computer Science, University of California, Riverside, June 2002.
[153] P. R. Pietzuch, J. Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS 2003),* ACM 2003.
[154] P.R. Pietzuch. Hermes: A scalable event-based middleware. Technical Report TR590, University of Cambridge, 2004.
[155] J. Bacon, K. Moody. Toward open, secure, widely distributed services. In *Communications of the ACM*, Volume 45, Issue 6, June 2002.
[156] M. Neovius. *Managing trust in P2P and collaborative computing*. Master Thesis, Abo Akademi, 2005.
[157] M. Ronkko. Stepwise Development of Hybrid Systems. Academic Dissertation, Abo Akademi, 2001.
[158] J. Coutaz, J. L. Crowley, S. Dobson, D. Garlan. Context is Key. In *Communications of the ACM*, Volume 48, Number 3, 2005.
[159] SystemC: http://www.systemc.org/
[160] J.R. Burch, E.M. Clarke, D.E. Long, K.L. MacMillan, D.L. Dill. Symbolic Model Checking for Sequential Circuit Verification. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4): 401-424, 1994.

PART II:
PUBLICATION REPRINTS

Paper I

Stepwise Development of Peer-to-Peer Systems

L. Yan, K. Sere

Available in: *Proceedings of the 6th International Workshop in Formal Methods (IWFM'03), July 2003. British Computer Society Press.*

Stepwise Development of Peer-to-Peer Systems

Lu Yan and Kaisa Sere
Turku Centre for Computer Science (TUCS) and
Department of Computer Science, Åbo Akademi University,
FIN-20520 Turku, Finland
{*Lu.Yan, Kaisa.Sere*}*@abo.fi*

Abstract

Peer-to-peer systems like Napster, Gnutella and Kazaa have recently become popular for sharing information. In this paper, we show how to design peer-to-peer systems within the action systems framework by combining UML diagrams. We present our approach via a case study of stepwise development of a Gnutella-like peer-to-peer system.

Keywords: stepwise, peer-to-peer, action systems, specification, Gnutella

1. INTRODUCTION

Peer-to-peer systems like Napster, Gnutella and Kazaa have recently become popular for sharing information. People find in peer-to-peer applications a convenient solution to exchange resources via internet. Two factors have fostered the recent explosive growth of such systems: first, the low cost and high availablity of large numbers of computing and storage resource, and second, increased network connectivity. As these trends continue, the peer-to-peer paradigm is bound to be more popular [9].

Most current distributed systems like the Web follow the client-server paradigm in which a single server stores information and distributes it to clients upon their requests. Peer-to-peer systems, which consider that all nodes are equal for sharing information, on the other hand, follow a paradigm in which each node can store information of its own and establish direct connections with another node to download information. In this way, the peer-to-peer systems offer attractive advantages like enhanced load balancing, dynamic information repositories, redundancy and fault tolerance, content-based addressing and improved searching [11] over the traditional client-server systems.

Because of the surprisingly rapid deployment of some peer-to-peer applications and the great advantages of the peer-to-peer paradigm, we are motivated to conduct a study of peer-to-peer systems and achieve a way to develop such systems. After using and analyzing various peer-to-peer clients on different platforms, we identified two common problems of clients: *reliability and robustness* (Most clients fail to provide satisfactory download service and some buggy ones even bring down the system during our test) and *extendability* (Most clients are implemented in a way that adding new services or functionalities results to lots of modifications to the original specifications). An attractive strategy to solve the first open problem is to use formal methods in designing peer-to-peer systems. Formal methods can help with reliability and robustness by minimizing errors in designing peer-to-peer systems. To improve extendability, we introduce a modular and object-oriented architecture for peer-to-peer systems. The benefit of object-orientation can be used to design and implement peer-to-peer systems in a reusable, composable and extendable way.

In this paper, we show how to design peer-to-peer systems within the action systems framework by combining UML diagrams. We present our approach via a case study of stepwise development of a Gnutella-like peer-to-peer system. We start by briefly describing the action systems framework to the required extent in Section 2. In Section 3 we give an initial specification of the Gnutella system. An abstract action system specification is derived in Section 4. In Section 5 we analyze and refine the system specification introduced in the previous section. We end in Section 6 with concluding remarks.

2. ACTION SYSTEMS

Action systems have proved to be very suitable for designing distributed systems [1, 2, 3, 6, 13]. The design and reasoning about action systems are carried out with refinement calculus [12].

In this section we will introduce OO-action systems [4], an object-oriented extension of action systems which we select as a foundation to work on. In this way, we can address our two open problems in a unified framework with benefits from both formal methods and object-orientation.

An OO-action system consists of a finite set of classes, each class specifying the behaviour of objects that are dynamically created and executed in parallel.

2.1. Actions

We will consider a fixed set *Attr* of attributes (variables) and assume that each attribute is associated with a nonempty set of *values*. Also, we consider a set *Act* of actions defined by the following grammar

$$A \ ::= \ abort|skip|x := v|x :\in V|b \rightarrow A|\{b\}|A;A|A \ [] \ A.$$

Here x is a list of attributes, v a list of values, V a nonempty set of values, b a predicate over attributes. Intuitively, $abort$ is the action which always deadlocks, $skip$ is a stuttering action, $x := v$ is a multiple assignment, $x :\in V$ is a random assignment, $b \rightarrow A$ is a guard of an action, $\{b\}$ is an assertion, $A_1; A_2$ is the sequential composition of the actions A_1 and A_2, and $A_1 \ [] \ A_2$ is the nondeterministic choice between the action A_1 and A_2.

The *guard* of an action is defined in a standard way using weakest preconditions [14]

$$g(A) \ = \ \neg wp(A, \textit{false}).$$

The action A is said to be enabled when the guard is true.

2.2. Classes and objects

Let *CName* be a fixed set of class names and *OName* a set of valid names for objects. We will also consider a fixed set of object variables *OVar* assumed to be disjoint from *Attr*. The only valid values for object variables are the names for objects in *OName*. The set of object actions *OAct* is defined by extending the grammar of actions as follows

$$\begin{array}{lll} O & ::= & A|n := o|new(c)|n := new(c) \\ & & |p|n.m|\textit{self.m}|super.m|O;O|O \ [] \ O. \end{array}$$

Here $A \in Act$, n is an object variable, o is either an object name or the constants *self* or $super$ (all three possibly resulting from the evaluation of an expression), c is a class name, p a procedure name, and m is a method name. Intuitively, $n := o$ stores the object name o into the object variable n, $new(c)$ creates a new object instance of the class c, $n := new(c)$ assigns the name of the newly created instance of the class c to the object variable n, p is a procedure call, $n.m$ is a call of the method m of the object the name of which is stored in the object variable n, *self.m* is a call of the method m declared in the same object, and $super.m$ is a call of the method m declared in the object that created the calling object. Note that method calls are always prefixed by an object variable or by the constant *self* or $super$.

We define the guard $gd(O)$ of an object action O to be the guard of the action in Act obtained by substituting every atomic object action of O with the action $skip$, where an atomic object action is

$$A, n := o, new(c), n := new(c), p, n.m, \textit{self.m}, super.m.$$

The resulting statement is an action in Act and hence its guard is well defined.

A $class$ is a pair $< c, \mathcal{C} >$, where $c \in CName$ is the $name$ of the class and $\mathcal{C}$ is its $body$, that is, a statement of the form

$$\begin{array}{lll} \mathcal{C} \;=\; |[& \mathbf{attr} & y^* := y0; x := x0 \\ & \mathbf{obj} & n \\ & \mathbf{meth} & m_1 = M_1; \cdots; m_h = M_h \\ & \mathbf{proc} & p_1 = P_1; \cdots; p_k = P_k \\ & \mathbf{do}\ O\ \mathbf{od} & \\ ]| & & \end{array}$$

A class body consists of an object action O and of four declaration sections. In the attribute declaration the *shared attributes* in the list y, marked with an asterisk $*$, describe the variables to be shared among all active objects. Therefore they can be used by instances of the class $\mathcal{C}$ and also by object instances of other classes. Initially they get values $y0$. The *local attributes* in the list x describe variables that are local to an object instance of the class, meaning that they can only be used by the instance itself. The variables are initialized to the value $x0$.

The list n of *object variables* describes a special kind of variables local to an object instance of the class. They contain names of objects and are used for calling methods of other objects. We assume that the lists x, y and n are pairwise disjoint.

A *method* $m_i = M_i$ describes a procedure of an object instance of the class. They can be called by actions of the objects themselves or by actions of another object instance of possibly another class. A method consists of a method name m and an object action M.

A *procedure* $p_i = P_i$ describes a procedure that is local to the object instances of the class. It can be called only by actions of the object itself. Like a method, it consists of a procedure name p and an object action forming the body P.

The *class body* is a description of the actions to be executed repeatedly when the object instance of the class is activated. It can refer to attributes which are declared to be shared in another class, and to the object variables and the local attributes declared within the class itself. It can contain procedure calls only to procedures declared in the class and method calls of the form $n.m$ or $super.m$ to methods declared in other classes. Method calls *self.m* are allowed only if m is a method declared in the same class. As for action systems, the execution of an object action is atomic.

2.3. OO-action system

An *OO-action system* OO consists of a finite set of classes

$$OO \;=\; \{< c_1, \mathcal{C}_1 >, \cdots, < c_n, \mathcal{C}_n >\}$$

such that the shared attributes declared in each $\mathcal{C}_i$ are distinct and actions in each $\mathcal{C}_i$ or bodies of methods and procedures declared in each $\mathcal{C}_i$ do not contain new statements referring to class names not used by classes in OO. Local attributes, object variables, methods, and procedures are not required to be distinct.

There are some classes in OO, marked with an asterisk $*$. Execution starts by the creation of one object instance of each of these classes. Each object, when created, chooses enabled actions and executes them. Actions operating on disjoint sets of local and shared attributes, and object variables can be executed in parallel. They can also create other objects. Actions of different active objects can be executed in parallel if they are operating on disjoint sets of shared attributes. Objects interact by means of the shared attributes and by executing methods of other objects.

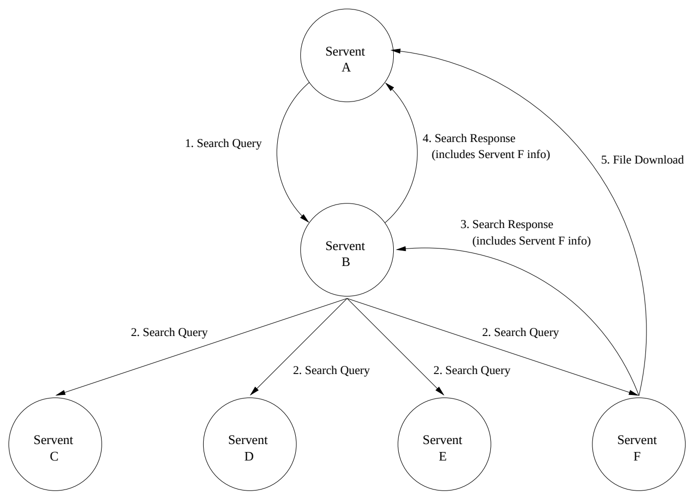


FIGURE 1: Gnutella peer-to-peer model [10]

3. INITIAL SPECIFICATION OF THE GNUTELLA SYSTEM

Gnutella is a decentralized peer-to-peer file-sharing model developed in 2000 by Nullsoft, AOL subsidiary and the company that created WinAMP [10]. The Gnutella model enables file sharing without using servers.

Unlike a centralized server network, the Gnutella network does not use a central server to keep track of all user files. To share files using the Gnutella model, a user starts with a networked computer A with a Gnutella *servent*, which works both as a server and a client. Computer A will connect to another Gnutella-networked computer B and then announce that it is *alive* to computer B. B will in turn announce to all its neighbours C, D, E, and F that A is alive. Those computers will recursively continue this pattern and announce to their neighbours that computer A is alive. Once computer A has announced that it is alive to the rest of the members of the peer-to-peer network, it can then search the contents of the shared directories of the peer-to-peer network.

Search requests are transmitted over the Gnutella network in a decentralized manner. One computer sends a search request to its neighbours, which in turn pass that request along to their neighbours, and so on. Figure 1 illustrates this model. The search request from computer A will be transmitted to all members of the peer-to-peer network, starting with computer B, then to C, D, E, F, which will in turn send the request to their neighbours, and so forth. If one of the computers in the peer-to-peer network, for example, computer F, has a match, it transmits the file information (name, location, etc.) back through all the computers in the pathway towards A (via computer B in this case). Computer A will then be able to open a direct connection with computer F and will be able to download that file directly from computer F.

4. ACTION SYSTEM SPECIFICATION OF THE GNUTELLA SYSTEM

When taking a step back, it is seen that the Gnutella system enables at least the following functionalities:

1. Servent can easily join and leave the peer-to-peer network.
2. Servent can publish its content to the shared directories of the peer-to-peer network.
3. Servent can search for and download files from the shared directories of the peer-to-peer network using keywords.

Based on the simple descriptions above, we can identify that servent should provide three basic services, *connect service*, *lookup service* and *download service*, as shown in Fig.2. From this

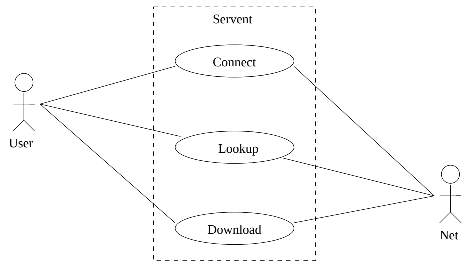


FIGURE 2: Use Case diagram of servent

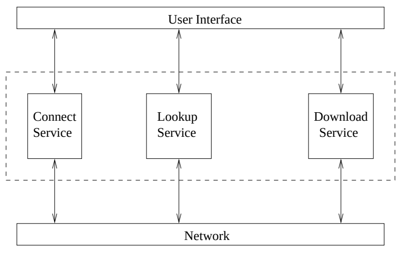


FIGURE 3: Structure diagram of servent

diagram we divide the system into components and derive a component-based structure of servent in Fig.3.

The statechart diagram Fig.4 shows the joint behaviour of servent. Each state is described by a set of attributes. We give unique preconditions for entering each state. Table 1 shows preconditions and invariants for every state of the servent.

An interesting issue to notice is that downloads can be initiated in two ways, i.e. either from a search result or by directly specifying target information. This design is reasonable because we do not always need to search the peer-to-peer network to get wanted files. In some cases, name and location information of files is already available. For example, file exchanges between two friends, who have already known each other's IP and shared contents. Take this into consideration, we provide two ways to initiate downloads.

The first version of action system specification of servent can be derived directly from Fig.4 and Table 1.

$$\{< GnutellaServent, GS >, \cdots\}$$

where the class body in Table 2 consists of attribute declaration, initialisation and a loop of actions which are chosen for execution in a non-deterministic fashion when enabled. Each action is of the form $g \rightarrow S$ where g is the guard and S is a statement to be executed when the guard evaluates to

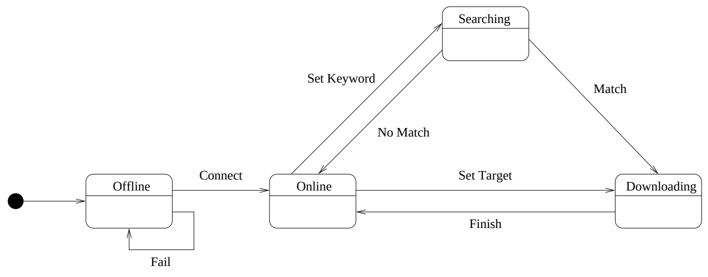


FIGURE 4: Statechart diagram of servent

TABLE 1: Preconditions and invariants for states

State	Precondition	Invariant
Offline	¬connected $\wedge$ keyword = ϕ $\wedge$ target = ϕ	keyword = ϕ $\wedge$ target = ϕ
Online	connected $\wedge$ keyword = ϕ $\wedge$ target = ϕ	connected
Searching	connected $\wedge$ keyword $\neq \phi$ $\wedge$ target = ϕ	connected $\wedge$ target = ϕ
Downloading	connected $\wedge$ keyword = ϕ $\wedge$ target $\neq \phi$	connected $\wedge$ keyword = ϕ

TABLE 2: Initial specification of servent

```
$GS \ = \ |[$ $\mathbf{attr}$  $connected^*; keyword^*; target^*; state := \textit{Offline}$
         $\mathbf{do}$
               $state = \textit{Offline} \wedge connected \rightarrow$
                   $state := Online$
             $[]\ state = Online \wedge keyword \neq \phi \rightarrow$
                   $state := Searching$
             $[]\ state = Online \wedge target \neq \phi \rightarrow$
                   $state := Downloading$
             $[]\ state = Searching \wedge target = \phi \rightarrow$
                   $keyword := \phi; state := Online$
             $[]\ state = Searching \wedge target \neq \phi \rightarrow$
                   $keyword := \phi; state := Downloading$
             $[]\ state = Downloading \rightarrow$
                   $target := \phi; state := Online$
         $\mathbf{od}$
       $]|$
```

$true$. Here *connected* is a boolean variable; *keyword* is the search criteria; *target* is the location information of shared resources in format *filename@IP*.

The next step is to apply the design in Fig.3 to our initial specification, which results in three more classes *ConnectService, LookupService* and *DownloadService*. Now the system consists of a set of classes $< c, C >$ where c is the name of the class and C is its body. On the top level, we have components of servent

$$\{< GnutellaServent, GS >^*, < ConnectService, Cs >,$$
$$< LookupService, Ls >, < DownloadService, Ds >, \cdots\}$$

The class $< GnutellaServent, GS >$, marked with an asterisk $*$, is the root class. At execution time one object of this class is initially created and this in turn creates other objects by executing new statements.

Let us look at the actions in Table 2. We can refine them according to service groups. For example, action

$$state = \textit{Offline} \wedge connected \rightarrow state := Online$$

where *Offline* and *Online* are defined in Table 1, can be refined into action

$$\neg connected \rightarrow connected := c.Connect(\ )$$

In this paper, however, we skip refinement details here because we do not want to go into details of semantics of action systems [15] nor refinement rules of refinement calculus [12].

TABLE 3: Refined specification of servent

```
GS = |[ attr  $connected := \text{false}; keyword := \phi; target := \phi$
        obj   $c : ConnectService; l : LookupService;$
              $d : DownloadService$
        meth  $SetKeyword(k) = keyword := k;$
              $SetTarget(t) = target := t$
        init  $c := new(ConnectService);$
              $l := new(LookupService);$
              $d := new(DownloadService)$
        do
                $\neg connected \rightarrow connected := c.Connect(\ )$
                $[]\ connected \wedge keyword \neq \phi \rightarrow$
                    $target := l.Search(keyword); keyword := \phi$
                $[]\ connected \wedge target \neq \phi \rightarrow$
                    $d.Download(target); target := \phi$
        od
     ]|
```

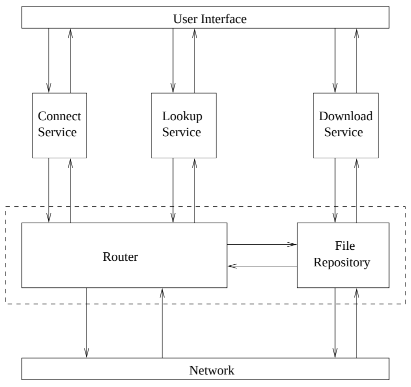


FIGURE 5: Schematic diagram of servent

The body of the refined specification of servent is described in Table 3. It models a servent that provides three basic services (*ConnectService, LookupService* and *DownloadService*). When it connects itself to the peer-to-peer network, users can search the network via $SetKeyword$ method and then download files from the search result. Or alternatively, users can directly give *target* information via $SetTarget$ method to download files.

5. REFINING GNUTELLA SERVENT

Ultimately, we need to derive an implementable specification for each service in the Guntella servent. Hence, every service component should be refined. We notice *ConnectService* and *LookupService* share a common functionality that enables appropriate message routing. It is reasonable to introduce a new component *Router* to the system as depicted in Fig. 5. This component will be in charge of routing all the incoming and outgoing messages of the servent. For *DownloadService*, we introduce a new component *FileRepository* in Fig. 5. It will act as a resource exchanger between servent and network.

5.1. Refining *ConnectService*

We start by considering *ConnectService* first. A Gnutella servent connects itself to the peer-to-peer network by establishing a connection with another servent currently on the network, and this

TABLE 4: Message format

$$
\begin{array}{lll}
Msg\ =\ |[& \textbf{attr} & descriptorID; TTL; type; \mathit{info} \\
 & \textbf{meth} & Transmit(\ ) = TTL > 0 \rightarrow TTL := TTL - 1 \\
 & \textbf{init} & t \in \{Ping, Pong\} \rightarrow \\
 & & \quad Msg(t) = (descriptorID := \_unique\_ID\_; \\
 & & \quad TTL := \_max\_TTL\_; type := t; \mathit{info} := \_\mathit{info}\_) \\
\quad]| & &
\end{array}
$$

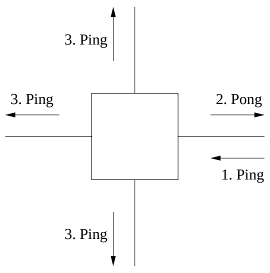


FIGURE 6: Ping - Pong routing[7]

kind of connection is passed around recursively. In order to model the communication between servents, we define a set of descriptors and inter-servent descriptor exchange rules as follows [7]

Ping Used to actively discover hosts on the network. A servent receiving a Ping descriptor is expected to respond with one or more Pong descriptors.

Pong The response to a Ping. Includes the address of a connected Gnutella servent and information regarding the amount of data it is making available to the network.

Furthermore, we need to define the format of Ping descriptor and Pong descriptor. We use the message format in Table 4, where *DescriptorID* is a string uniquely identifying the descriptors on the network. TTL stands for *Time To Live*, which is the number of times the descriptor will be forwarded by servent before it is removed from the network. The TTL is the only mechanism for expiring descriptors on the network. Each servent will decrement the TTL before passing it on to another servent. When the TTL reaches 0, the descriptor will no longer be forwarded. The information carried by the message is encoded in *info*, whose format depends on the variable *type*.

The peer-to-peer nature of the Gnutella network requires servents to route network traffic appropriately. Intuitively, a servent will forward incoming Ping descriptors to all its directly connected servents. Pong descriptors may only be sent along the same path that carried the incoming Ping descriptor as shown in Fig.6. This ensures that only servents that routed the Ping descriptors will see the Pong descriptor in response. A servent that receives a Pong descriptor with $descriptorID = n$, but has not seen a Ping descriptor with $descriptorID = n$ should remove the Pong descriptor from the network.

The above routing rules can be illustrated in the statechart diagram Fig. 7. Using the same techniques as in the previous section, we can translate the diagram into action system specification and further refine it into Table 5.

The specification of Ping - Pong router models a router that can route Ping - Pong traffic appropriately. When the router is initiating, it connects itself to the peer-to-peer network by sending Ping descriptors to other peers. After initiation, it continues receiving *_incoming_message_* and replying with apporiate *_outgoing_message_*. Here *descriptorDB* is a set storing *descriptorID*

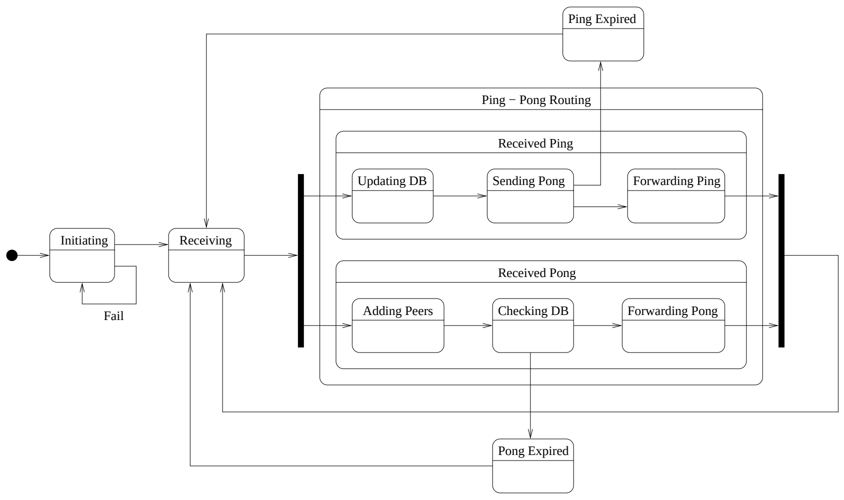


FIGURE 7: Statechart diagram of Ping - Pong routing rules

TABLE 5: Specification of Ping - Pong router

```
$Rc\ =\ |[$  **attr**  $connected := \textsf{false}; descriptorDB := \phi; peers := \phi$
       **obj**   $receivedMsg : Msg; newMsg : Msg$
       **meth**  $ReceiveMsg(\ ) = receivedMsg := \_incoming\_message\_;$
             $SendPing(\ ) = (newMsg := new(Msg(Ping));$
                 $\_outgoing\_message\_ := newMsg);$
             $SendPong(\ ) = (newMsg := new(Msg(Pong));$
                 $newMsg.\textsf{info}.IP := \_this\_IP\_;$
                 $\_outgoing\_message\_ := newMsg);$
             $\textsf{ForwardMsg}(m) = (m.TTL > 0 \rightarrow$
                 $m.Transmit(\ ); \_outgoing\_message\_ := m)$
       **init**  $SendPing(\ )$
       **do**
              $|[\ peers \neq \phi \rightarrow connected := true$
              $[]\ \neg connected \rightarrow SendPing(\ )\ ]|$
              $/\!/$
              $|[\ true \rightarrow$
                  $ReceiveMsg(\ );$
                  **if** $receivedMsg.type = Ping \rightarrow$
                      $descriptorDB := descriptorDB \cup$
                      $receivedMsg.descriptorID;$
                      $SendPong(\ );$
                      $\textsf{ForwardMsg}(receivedMsg)$
                  $[]\ receivedMsg.type = Pong \rightarrow$
                      $peers := peers \cup receivedMsg.\textsf{info}.IP;$
                      $receivedMsg.descriptorID \in descriptorDB \rightarrow$
                          $\textsf{ForwardMsg}(receivedMsg)$
                  **fi**
              $]|$
       **od**
   $]|$
```

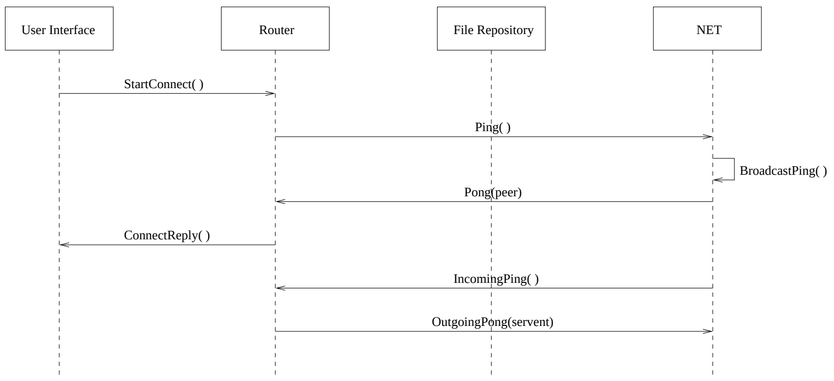


FIGURE 8: Sequence diagram of a connect session

TABLE 6: Specification of *ConnectService*

$$
\begin{array}{lll}
Cs = |[& \mathbf{attr} & connected := \textit{false} \\
 & \mathbf{obj} & r : Router \\
 & \mathbf{meth} & Connect(\,) = (connected := r.connected) \\
 & \mathbf{init} & r := new(Router) \\
]| & &
\end{array}
$$

information; *peers* is a set storing its directly and indirectly connected servents information; and *_this_IP_* is the IP address of the responding servent. The sequence of a connect session is summarized in Fig. 8.

In order to reuse the specification in Table 3, we will specify *ConnectService* without making any changes in its interface. The specification is shown in Table 6. When *ConnectService* is initiating, an instance of *Router* is created. Then it keeps checking state variable *connected* in the router and passing the status to servent.

5.2. Refining *LookupService*

When we think about *LookupService*, we follow almost the same paradigm as *ConnectService* to specify this component. A Gnutella servent starts a search request by broadcasting a *Query* message through the peer-to-peer network. Upon receiving a search request, the servent checks if any local stored files match the query and sends a *QueryHit* message back. We use following descriptors and routing rules to model the communication between servents [7]

Query The primary mechanism for searching the distributed network. A servent receiving a Query descriptor will respond with a QueryHit if a match is found against its local data set.

QueryHit The response to a Query. This descriptor provides the recipient with enough information to acquire the data matching the corresponding Query.

The message format in Table 4 has to be revised to adopt the new descriptors. The message *type* now includes *Ping, Pong, Query* and *QueryHit*, so a minor change is made in Table 7.

The routing rules for Query - QueryHit traffic are similar to the rules for Ping - Pong traffic. A servent will forward incoming Query descriptors to all its directly connected servents. QueryHit descriptors may only be sent along the same path that carried the incoming Query descriptor as shown in Fig. 9. This ensures that only those servents that routed the query descriptors will see the QueryHit descriptor in response. A servent that receives a QueryHit descriptor with

TABLE 7: Refined message format

$$
\begin{array}{lll}
Msg = |[& \mathbf{attr} & descriptorID; TTL; type; \textit{info} \\
 & \mathbf{meth} & Transmit(\,) = TTL > 0 \rightarrow TTL := TTL - 1 \\
 & \mathbf{init} & t \in \{Ping, Pong, Query, QueryHit\} \rightarrow \\
 & & \quad Msg(t) = (descriptorID := \_unique\_ID\_; \\
 & & \quad TTL := \_max\_TTL\_; type := t; \textit{info} := \_\textit{info}\_) \\
]| & &
\end{array}
$$

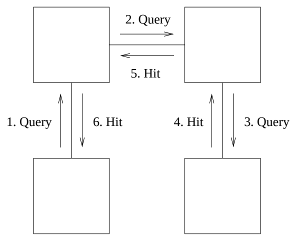


FIGURE 9: Query - QueryHit routing[7]

$descriptorID = n$, but has not seen a Query descriptor with $descriptorID = n$ should remove the QueryHit descriptor from the network.

Like the previous section, we first draw a statechart diagram for the Query - QueryHit routing rules. Then we translate it into action system specification and further refine it.

In Table 8 we have the body of Query - QueryHit router specification, which models a router that is in charge of routing Query - QueryHit traffic appropriately. Like a Ping - Pong router, it keeps receiving *_incoming_message_* and replying with apporiate *_outgoing_message_*. Here *descriptorDB* is a set storing *descriptorID* information; *myKeyword* is a string storing search criteria; *cKeyword* is a string storing comparison criteria; *filename* is a string storing destination filename; *target* is the shared resource location information in format *filename@IP*; and *f* is an object of class *FileRepository* which enables local file search service via *Has* and *Find* methods. Details of class *FileRepository* will be elaborated in the next section.

The Query - QueryHit router provides searching function via method *SetKeyword*. Once a keyword is set, a Query descriptor carrying search criteria is generated and broadcasted in the peer-to-peer network via method *SendQuery*. In the mean time, the router keeps receiving Query and QueryHit descriptors. For an incoming Query descriptor, a query request is passed to *FileRepository*. According to the search result, a QueryHit descriptor is sent back in response via method *SendQueryHit* if a match is found, otherwise the Query descriptor is further forwarded via method *ForwardMsg*. Upon receiving a QueryHit descriptor, it checks its keyword field, and then sets *target* information to complete the search session. We summarize the sequence of a search session in Fig. 10.

Now we specify *LookupService* with emphasis on specification reuse. The result is shown in Table 9. When *LookupService* is initiated, an instance of *Router* is created. It provides *Search* method via calling *SetKeyword* method in the router, and then returning the search result to servent.

Until now we have two action systems, *Rc* modeling Ping - Pong routing rules and *Rl* modeling Query - QueryHit routing rules. We notice the two action systems actually model different aspects of a full router. Furthermore, we can compose the two action systems together using *prioritising composition* [5] to derive the action system specification of a full router

$$R = |[\ Rc \,/\!/\, Rl\ ]|$$

TABLE 8: Specification of Query - QueryHit router

```
$Rl$ = |[ attr  $descriptorDB := \phi; myKeyword := \phi;$
                $cKeyword := \phi;$ filename $:= \phi; target := \phi$
          obj   $receivedMsg : Msg; newMsg : Msg; f :$ FileRepository
          meth  $SetKeyword(k) = myKeyword := k;$
                $ReceiveMsg(\ ) = receivedMsg := \_incoming\_message\_;$
                $SendQuery(\ ) = (newMsg := new(Msg(Query));$
                    $newMsg.$info$.keyword := myKeyword;$
                    $cKeyword := myKeyword; target := \phi;$
                    $\_outgoing\_message\_ := newMsg);$
                $SendQueryHit(\ ) = (newMsg := new(Msg(QueryHit));$
                    $newMsg.$info$.keyword := receivedMsg.$info$.keyword;$
                    $newMsg.$info.filename $:=$ filename;
                    $newMsg.$info$.IP := \_this\_IP\_;$
                    $\_outgoing\_message\_ := newMsg);$
                ForwardMsg$(m) = (m.TTL > 0 \rightarrow$
                    $m.Transmit(\ ); \_outgoing\_message\_ := m)$
          do
                 |[ $myKeyword \neq \phi \rightarrow SendQuery(\ ); myKeyword = \phi$ ]|
                 //
                 |[ $true \rightarrow$
                      $ReceiveMsg(\ );$
                      if $receivedMsg.type = Query \rightarrow$
                          $descriptorDB := descriptorDB \cup$
                          $receivedMsg.descriptorID;$
                          if $f.Has(receivedMsg.$info$.keyword) \rightarrow$
                              filename $:= f.Find(receivedMsg.$info$.keyword);$
                              $SendQueryHit(\ )$
                          [] $\neg f.Has(receivedMsg.$info$.keyword) \rightarrow$
                              ForwardMsg$(receivedMsg)$
                          fi
                      [] $receivedMsg.type = QueryHit \rightarrow$
                          if $receivedMsg.$info$.keyword = cKeyword \rightarrow$
                              $target := receivedMsg.$info.filename@
                              $receivedMsg.$info$.IP; cKeyword := \phi$
                          [] $receivedMsg.$info$.keyword \neq cKeyword \wedge$
                          $receivedMsg.descriptorID \in descriptorDB \rightarrow$
                                  ForwardMsg$(receivedMsg)$
                          fi
                      fi
                 ]|
          od
      ]|
```

TABLE 9: Specification of *LookupService*

```
$Ls$ = |[ attr  $target := \phi$
          obj   $r : Router$
          meth  $Search(k) = (r.SetKeyword(k); target := r.target)$
          init  $r := new(Router)$
      ]|
```

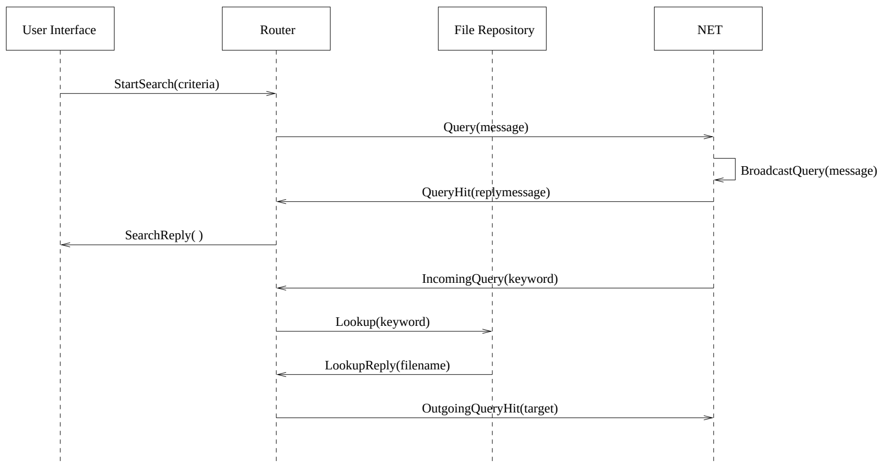


FIGURE 10: Sequence diagram of a search session

where on the higher level, we have components of the router

$$\{< Router, R >, < PingPongRouter, Rc >, < QueryRouter, Rl >\}$$

5.3. Refining *DownloadService*

DownloadService is relatively simple compared to *ConnectService* and *LookupService*. The primary function of this component is to enable a servent to download files from other servents. Once a servent receives a QueryHit descriptor, it may initiate the direct download of one of the files described by the descriptor's result set. Or alternatively, users can initiate the download directly by giving complete *target* information. Files are downloaded out-of-network, i.e. a direct connection between the source and target servent is established in order to perform the data transfer. File data is never transferred over the peer-to-peer network.

Additionally, this component provides the local file query function for other servents. It should be in charge of a local file database which provides data services like *add, delete, update, query* and *refresh* etc. Moreover, it should take full control of local files. Hence we introduce a new component *FileRepository* in Table 10, which will satisfy the above requirements for *DownloadService*. First of all, we provide *SetTarget* method to enable file downloads. To make things simple, we assume that *fileDB* is simply a set of relations $\{key\} \rightarrow \{\mathit{file}\}$. We use relation notations [16] *dom* and *ran* for domain and range operations, and $\lhd$ as a domain restriction operator, defined by $S \lhd r = \{x, y | x \mapsto y \in r \land x \in S\}$. For incoming Query descriptors, *Has* and *Find* methods are provided to enable local file searches.

Given *target* information, download action is enabled and servent initiates a download. A download request is sent to the target servent, and then a file is downloaded via HTTP protocol. Afterwards, *fileDB* is refreshed in order to reflect the change of adding new files to the repository. The sequence of a download session is summarized in Fig. 11.

The last step is to specify *DownloadService*. From the result in Table 11, we can see that when *DownloadService* is initiating, an instance of *FileRepository* is created. It enables *Download* by calling *SetTarget* method in *FileRepository*.

At this stage of the design, we finally have a complete set of classes which are refinement results from the initial specification of servent as follows

$$\begin{gathered}\{< GnutellaServent, GS >^*, < ConnectService, Cs >,\\ < LookupService, Ls >, < DownloadService, Ds >,\\ < PingPongRouter, Rc >, < QueryRouter, Rl >,\\ < Router, R >, < \text{FileRepository}, F >, < Message, Msg >\}\end{gathered}$$

TABLE 10: Specification of file repository

```
F = |[ attr  $\mathit{fileDB} := \_\mathit{fileDB}\_; \mathit{filename} := \phi; target := \phi$
       meth  $SetTarget(t) = (target := t);$
             $Has(key) = (\{key\} \in dom(\mathit{fileDB}));$
             $Find(key) = (\mathit{filename} := \mathit{file} \wedge \{\mathit{file}\} \in ran(\{key\} \lhd \mathit{fileDB}))$
       do
             $target \neq \phi \rightarrow$
                 $HTTP(target);$
                 $target := \phi;$
                 $\mathit{Refresh}(\mathit{fileDB})$
       od
    ]|
```

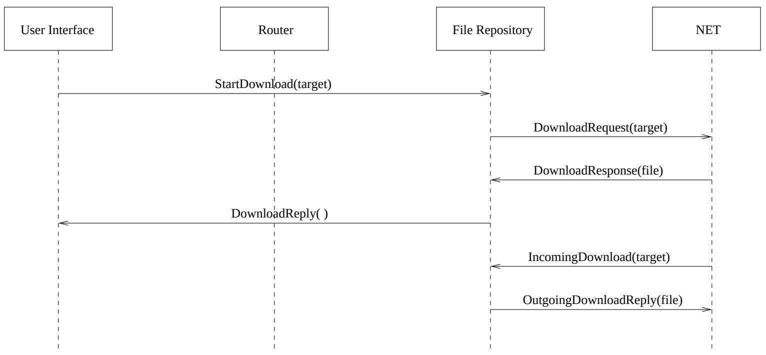


FIGURE 11: Sequence diagram of a download session

TABLE 11: Specification of *DownloadService*

$$
\begin{array}{lll}
Ds \ = \ |[& \mathbf{obj} & f : \textit{FileRepository} \\
 & \mathbf{meth} & Download(t) = f.SetKeyword(t) \\
 & \mathbf{init} & f := new(\textit{FileRepository}) \\
]| & &
\end{array}
$$

6. CONCLUDING REMARKS

We identified two open problems of existing peer-to-peer systems: *reliability and robustness* and *extendability*, and proposed strategies that can be used to solve them. The main contribution of this paper is an approach to stepwise development of peer-to-peer systems within the action systems framework by combining UML diagrams. We have presented our approach via a case study of stepwise development of a Gnutella-like peer-to-peer system.

Our experience shows that it is beneficial to combine informal methods like UML and formal methods like action systems together in the development of peer-to-peer systems. In the early stage, we try to catch the characteristic of the system using use case diagrams and statechart diagrams. Then formal specification in action systems framework is derived by further studying and elaborating details of these diagrams. In the later stage, sequence diagrams are used to graphically clarify the structure of the refined action system specification.

Moreover, we find OO-action systems very suitable for designing such kind of systems. The formal nature of OO-action systems makes it a good tool to built reliable and robust systems. Meanwhile, the object-oriented aspect of OO-action systems helps to built systems in an extendable way, which will generally ease and accelerate the design and implementation of new services or functionalities. Furthermore, the final set of classes in the OO-action system specification is easy to be implemented in popular OO-languages like Java, C++ or C#.

Peer-to-peer systems have been evolving very quickly. Besides Gnutella, another promising choice is JXTA [8] from Sun, which has been generating lots of attention. In the future work, we plan to further investigate this new standard. Moreover, we plan to stepwise implement our action system specification and develop it into a real peer-to-peer system.

REFERENCES

[1] R.J.R. Back and K. Sere: *From Action Systems to Modular Systems.* Software - Concepts and Tools. (1996) 17: 26–39.

[2] R.J.R. Back, A.J. Martin and K.Sere: *Specifying the Caltech asynchronous microprocessor.* Science of Computer Programming. (1996) 26: 79–97.

[3] L. Petre, M. Qvist and K. Sere: *Distributed Object-Based Control Systems.* TUCS Technical Reports, No 241, February 1999.

[4] M. Bonsangue, J.N. Kok and K. Sere: *An approach to object-orientation in action systems.* Proceedings of Mathematics of Program Construction (MPC'98), Marstrand, Sweden, June 1998. Lecture Notes in Computer Science 1422. Springer Verlag.

[5] E. Sekerinski and K. Sere: *A Theory of Prioritising Composition.* TUCS Technical Reports, No 5, May 1996.

[6] M. Butler, E. Sekerinski and K. Sere: *An Action System Approach to the Steam Boiler Problem.* In Jean-Raymond Abrial, Egon Borger and Hans Langmaack, editors, Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control. Lecture notes in Computer Science Vol. 1165. Springer-Verlag, 1996.

[7] Clip2 DSS: *Gnutella Protocol Specification v0.4.*
Online. http://www.clip2.com/GnutellaProtocol04.pdf.

[8] L. Gong: *JXTA: A network programming environment.* IEEE Internet Computing, 5(3): 88–95, May/June 2001.

[9] M. Ripeanu: *Peer-to-peer architecture case study: Gnutella network.* Technical Report TR-2001-26, University of Chicago, Department of Computer Science, July 2001.

[10] I. Ivkovic: *Improving Gnutella Protocol: Protocol Analysis and Research Proposals.* Technical report, LimeWire LLC, 2001.

[11] M. Parameswaran, A. Susarla and A.B. Whinston: *P2P networking: An information-sharing alternative.* IEEE Computer, 34(7): 31–38, July 2001.

[12] R.J. Back and J. Wright: *Refinement Calculus: A Systematic Introduction.* Graduate Texts in Computer Science, Springer-Verlag, 1998.

[13] L. Petre and K. Sere: *Coordination Among Mobile Objects.* Proceeding of IFIP TC6/WG6 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99), Florence, Italy, February 1999.

[14] E.W. Dijkstra: *A Discipline of Programming.* Prentice-Hall International, 1976.

[15] K. Sere: *Stepwise derivation of parallel algorithms.* Academic dissertation of Åbo Akademi Department of Computer Science, 1990.

[16] E. Sekerinski and K. Sere (Eds): *Program Development by Refinement: Case Studies Using the B Method.* Springer-Verlag, 1999.

Paper II

Via Firewalls

L. Yan

Available in: *Proceedings of the 3rd International Conference on Grid and Cooperative Computing (GCC'04), October 2004. Lecture Notes in Computer Science 3252, Springer-Verlag.*

Via Firewalls

Lu Yan

Turku Centre for Computer Science (TUCS) and
Department of Computer Science, Åbo Akademi University,
FIN-20520 Turku, Finland.
Lu.Yan@abo.fi

Abstract. A lot of networks today are behind firewalls. In peer-to-peer networking, firewall-protected peers may have to communicate with peers outside the firewall. This paper shows how to design peer-to-peer systems to work with different kinds of firewalls within the object-oriented action systems framework by combining formal and informal methods. We present our approach via a case study of extending a Gnutella-style peer-to-peer system to provide connectivity through firewalls.

1 Introduction

The idea of peer-to-peer networking, in a sense that nodes on the network communicate directly with each other, is as old as Internet itself. Internet used to be a peer-to-peer network if we go back to those early days in the 70's when Internet was limited to researchers in a few selected laboratories. Nowadays Internet has developed into a non peer-to-peer network, in the sense that most exchanges rely on mediation through gateways and servers. Moreover, most networks today employ firewalls, for security reasons, which impede direct communication by filtering packets and limiting the port numbers open to bi-directional traffic.

The goal of peer-to-peer networking is to remove the distinction between client and server. Instead of running web browers that can only request information from web server, users can run peer-to-peer applications to contribute contents or resources in addition to requesting them. As a vision of peer-to-peer networking, it is necessary for peer-to-peer applications to work in most environments, whether home, small business, or enterprise.

Previously [1], we have specified a Gnutella-like peer-to-peer system within the object-oriented action systems framework by combining UML diagrams. When implementing such a system in Java, we realized that a lot of networks today are behind firewalls. In peer-to-peer networking, firewall-protected peers may have to communicate with peers outside the firewall. Thus a solution should be made to create communication schemes that overcome the obstacles placed by the firewalls to provide universal connectivity throughout the network. This motivates us to conduct a study of firewalls in peer-to-peer networking and achieve a way to traverse firewalls. We present our approach via a case study of extending a Gnutella-style peer-to-peer system to provide connectivity through firewalls.

This paper shows how to design peer-to-peer systems to work with different kinds of firewalls within the object-oriented action systems framework by combining formal and informal methods. Formal methods can help with reliability and robustness by minimizing errors in designing peer-to-peer systems and the benefit of object-orientation can be used to design and implement peer-to-peer systems in a reusable, composable and extenable way.

As firewalls have various topologies (single, double, nested, etc.) and various security policies (packet filtering, one-way-only, port limiting, etc.), our problem has multiple faces and applications have multitude requirements. A general solution that fits all situations seems to be infeasible in this case. Thus we define a simplified version of the problem as shown in Fig. 1: How to provide connectivity between a private peer and a public peer through a single firewall?

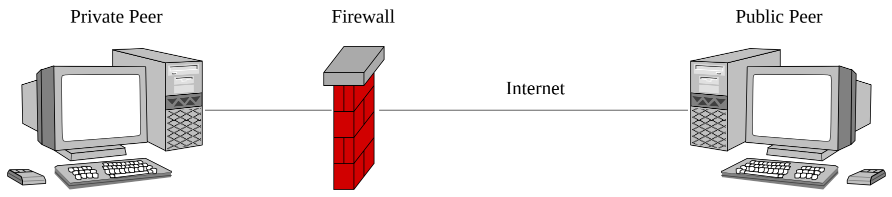


Fig. 1. Problem definition

The rest of the paper is organized as follows. A solution to uni-directional firewalls is derived in Section 2. In Section 3 we provide a solution to another kind of firewalls which limit the open port numbers. We end in Section 4 with related work and concluding remarks.

2 Uni-directional firewalls

Most corporate networks today are configurated to allow outbound connections (from the firewall protected network to Internet), but deny inbound connections (from Internet to the firewall protected network) as illustrated in Fig. 2.

These corporate firewalls examine the packets of information sent at the transport level to determine whether a particular packet should be blocked. Each packet is either forwarded or blocked based on a set of rules defined by the firewall administrator. With packet-filering rules, firewalls can easily track the direction in which a TCP connection is initiated. The first packets of the TCP three-way handshake are uniquely identified by the flags they contain, and firewall rules can use this information to ensure that certain connections are initiate in only one direction. A common configuration for these firewalls is to allow all connections initiated by computers inside the firewall, and restrict all connections for computers outside the firewall. For example, firewall rules might

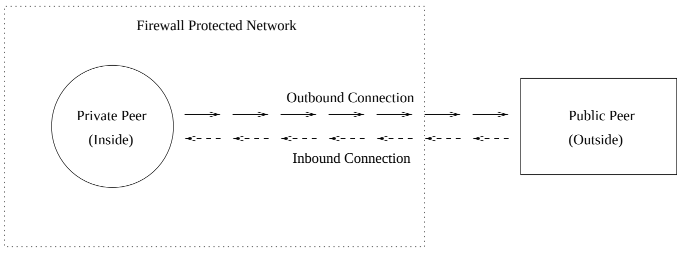


Fig. 2. Uni-directional Firewall

specify that users can browse from their computers to a web server on Internet, but an outside user on Internet cannot browse to the protected user's computer.

In order to traverse this kind of firewalls, we introduce a new descriptor and routing rules for servents [2]

Push A mechanism that allows a firewalled servent to contribute file-based data to the network. A servent may send a Push descriptor if it receives a QueryHit descriptor from a servent that doesn't support incoming connections.

The message format [1] has to be revised to adopt the new descriptor. The message *type* now includes *Ping, Pong, Query, QueryHit* and *Push*, so minor changes are made in Table 1.

Table 1. Message format

$$
\begin{array}{lll}
Msg = & |[\ \textbf{attr} & descriptorID; serventID; TTL; type; info \\
 & \quad \textbf{meth} & Transmit(\,) = TTL > 0 \rightarrow TTL := TTL - 1 \\
 & \quad \textbf{init} & t \in \{Ping, Pong, Query, QueryHit, Push\} \rightarrow \\
 & & \qquad Msg(t) = (descriptorID := \_unique\_ID\_; \\
 & & \qquad TTL := \_max\_TTL\_; type := t; info := \_info\_) \\
 &]| &
\end{array}
$$

Once a servent receives a QueryHit descriptor, it may initiate a direct download, but it is impossible to establish the direct connection if the servent is behind a firewall that does not permit incoming connections to its Gnutella port. If this direct connection cannot be established, the servent attempting the file download may request that the servent sharing the file *Push* the file instead. i.e. A servent may send a Push descriptor if it receives a QueryHit descriptor from a servent that doesn't support incoming connections.

Unlike the previous descriptors Ping, Pong, Query and QueryHit, Push descriptors are routed by *ServentID*, not by *DescriptorID*. Intuitively, Push descriptors may only be sent along the same path that carried the incoming QueryHit descriptors as illustrated in Fig. 3. This ensures that only those servents that routed the QueryHit descriptors will see the Push descriptor. A servent that receives a Push descriptor with $ServentID = n$, but has not seen a QueryHit descriptor with $ServentID = n$ should remove the Push descriptor from the network.

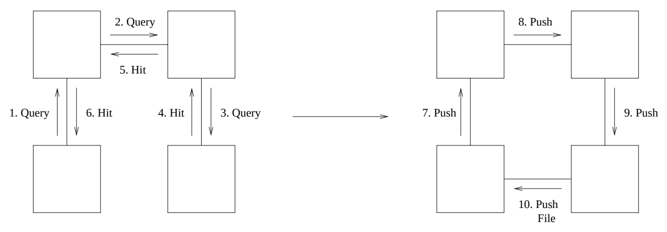


Fig. 3. Push routing [2]

We adopt uni-directional firewalls by adding a Push router in Table 2, which is an action system *Rf* modeling Push routing rules. Since this action system actually models a particular aspect of a full router, we can compose it with the previous two action systems *Rc* modeling Ping - Pong routing rules and *Rl* modeling Query - QueryHit routing rules together, using *prioritising composition* [3] to derive a new action system specification of a full router

$$R \;=\; |[\; Rc \,/\!/\, Rl \,/\!/\, Rf \;]|$$

where on the higher level, we have components of the router

$$\{< Router, R >, < PingPongRouter, Rc >,$$
$$< QueryRouter, Rl >, < PushRouter, Rf >\}$$

A servent can request a file push by routing a Push request back to the servent that sent the QueryHit descriptor describing the target file. The servent that is the target of the Push request should, upon receipt of the Push descriptor, attempt to establish a new TCP/IP connection to the requesting servent. As specified in the refined file repository in Table 3, when the direct connection is established, the firewalled servent should immediately send a HTTP GIV request with *requestIP*, *filename* and *destinationIP* information, where *requestIP* and *destinationIP* are IP address information of the firewalled servent and the target servent for the Push request, and *filename* is the requested file information. In this way, the initial TCP/IP connection becomes an outbound one, which is allowed by uni-directional firewalls. Receiving the HTTP GIV request, the target servent should extract the *requestIP* and *filename* information and construct an HTTP GET request with the above information. After that, the file download process is identical to the normal file download process without firewalls. We summarize the sequence of a Push session in Fig. 4.

Table 2. Specification of Push router

```
$Rf$ = |[ attr  $serventDB := \phi; cKeyword := \phi; filename := \phi;$
                $target := \phi; pushTarget := \phi;$
          obj   $receivedMsg : Msg; newMsg : Msg; f : FileRepository$
          meth  $SendPush(\,) = (newMsg := new(Msg(Push));$
                    $newMsg.info.requestIP := \_this\_IP\_;$
                    $newMsg.info.filename := receivedMsg.info.filename;$
                    $newMsg.info.destinationIP := receivedMsg.info.IP;$
                    $\_outgoing\_message\_ := newMsg);$
                $ReceiveMsg(\,) = receivedMsg := \_incoming\_message\_;$
                $ForwardMsg(m) = (m.TTL > 0 \rightarrow$
                    $m.Transmit(\,); \_outgoing\_message\_ := m)$
          do
                $true \rightarrow$
                    $ReceiveMsg(\,);$
                    if $receivedMsg.type = QueryHit \rightarrow$
                        $serventDB := serventDB \cup receivedMsg.serventID;$
                        if $receivedMsg.info.keyword = cKeyword \rightarrow$
                            $target := receivedMsg.info.filename@$
                                $receivedMsg.info.IP;$
                            if $f.firewall \rightarrow$
                                $SendPush(\,)$
                            fi
                            $cKeyword := \phi$
                        [] $receivedMsg.info.keyword \neq cKeyword \wedge$
                            $receivedMsg.descriptorID \in descriptorDB \rightarrow$
                                $ForwardMsg(receivedMsg)$
                        fi
                    [] $receivedMsg.type = Push \rightarrow$
                        if $receivedMsg.info.destinationIP = \_this\_IP\_ \rightarrow$
                            $pushTarget := receivedMsg.info.requestIP$®
                                $receivedMsg.info.filename@$
                                $receivedMsg.info.destinationIP$
                        [] $receivedMsg.info.destinationIP \neq \_this\_IP\_ \wedge$
                            $receivedMsg.serventID \in serventDB \rightarrow$
                                $ForwardMsg(receivedMsg)$
                        fi
                    fi
          od
        ]|
```

Table 3. Specification of file repository

```
$F$ = |[ attr  $firewall^* := false; fileDB := \_fileDB\_; cFileDB;$
               $filename := \phi; target := \phi; pushTarget := \phi$
         meth  $SetTarget(t) = (target := t);$
               $PushTarget(t) = (pushTarget := t);$
               $Has(key) = (\{key\} \in dom(fileDB));$
               $Find(key) = (filename := file \wedge \{file\} \in ran(\{key\} \lhd fileDB))$
         do
               $target \neq \phi \rightarrow$
                   $cFileDB := fileDB;$
                   $HTTP\_GET(target);$
                   $target := \phi;$
                   $Refresh(fileDB);$
                   if $fileDB = cFileDB \rightarrow$
                       $firewall := true$
                   [] $fileDB \neq cFileDB \rightarrow$
                       $firewall := false$
                   fi
               [] $pushTarget \neq \phi \rightarrow$
                   $HTTP\_GIV(pushTarget);$
                   $pushTarget := \phi;$
                   $Refresh(fileDB)$
         od
       ]|
```

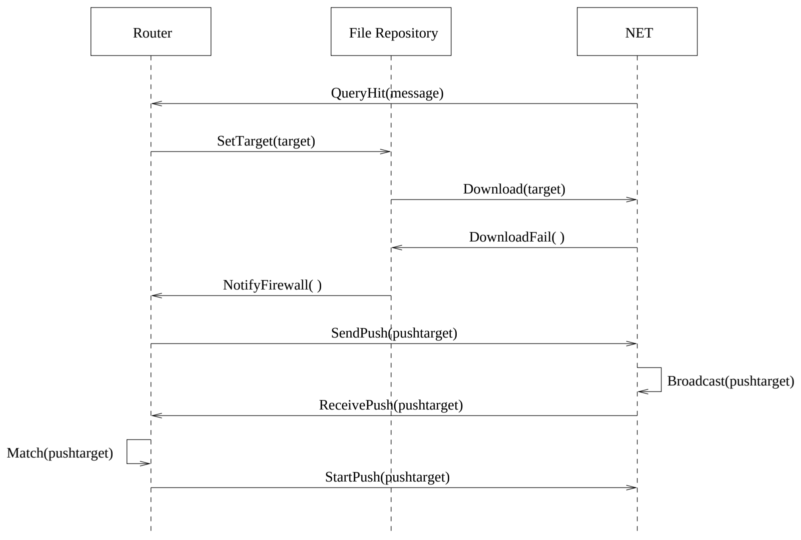


Fig. 4. Sequence diagram of a Push session

3 Port-blocking firewalls

In corporate networks, another kind of common firewalls are port-blocking firewalls, which usually do not grant long-time and trusted privileges to ports and protocols other than port 80 and HTTP/HTTPS. For example, port 21 (standard FTP access) and port 23 (standard Telnet access) are usually blocked and applications are denied network traffic through these ports. In this case, HTTP (port 80) has become the only entry mechanism to the corporate network. Using HTTP protocol, for a servent to communicate with another servent through port-blocking firewalls, the servent has to *pretend* that it is an HTTP server, serving WWW documents. In other words, it is going to mimick an *httpd* program.

When it is impossible to establish an IP connection through a firewall, two servents that need to talk directly to each other, solve this problem by having SOCKS support built into them, and having SOCKS proxy running on both sides. As illustrated in Fig. 5, it builds an HTTP-tunnel between the two servents.

After initializtion, the SOCKS proxy creates a *ProxySocket* and starts accepting connections on the Gnutella port. All the information to be sent by the attempting servent is formatted as a URL message (using the GET method of HTTP) and a *URLConnection* via HTTP protocol (port 80) is made. On the other side, the target servent accepts the request and a connection is establish with the attempting servent (actually with the SOCKS proxy in the target servent). The SOCKS proxy in the target servent can read the information sent by the attempting servent and write back to it. In this way, transactions between two servents are enabled.

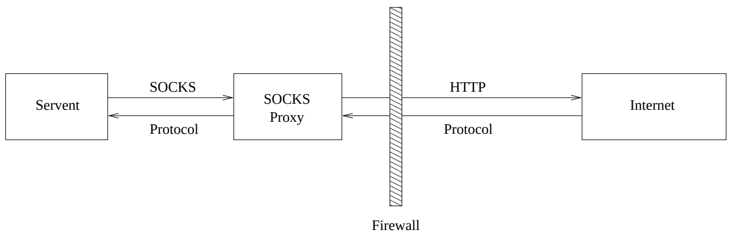


Fig. 5. Firewall architecture and extendable socket

We adopt port-blocking firewalls by adding a new layer to the architecture of servent in Fig. 6. This layer will act as a tunnel between servent and internet. As specified in Table 4, after receiving messages from the attemping servent and encoding them into HTTP format, the SOCKS proxy sends the messages to internet via port 80. In the reverse way, the SOCKS proxy keeps receiving messages from HTTP port and decoding them into original format. With this additional layer, our system can traverse port-blocking firewalls without any changes in its core parts. We summarize the sequence of a SOCKS proxy session in Fig. 7.

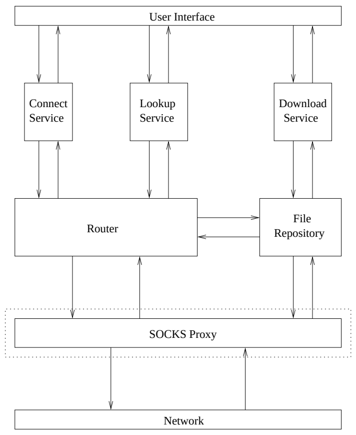


Fig. 6. Refined architecture of servent

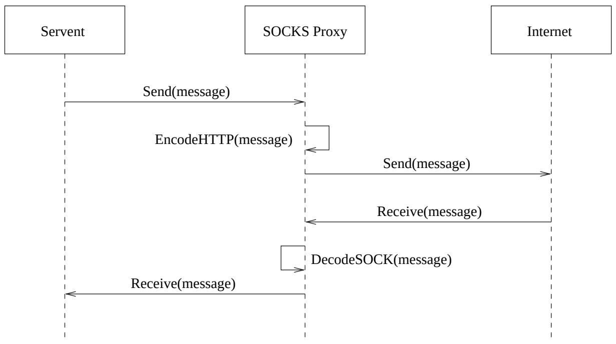


Fig. 7. Sequence diagram of a SOCKS proxy session

4 Related work and concluding remarks

The corporate firewall is a double-edged sword. It helps prevent unauthorized access to the corporate Web, but may disable access for legitimate peer-to-peer applications. JXTA [4] from Sun has provided an alternative solution to

Table 4. Specification of SOCKS proxy

```
S = |[ attr $listenPort := \_Gnutella\_port\_$;
            $destinationPort := 80$
       obj  $ProxySocket : Socket$;
            $HTTPSocket : Socket$;
            $imsg : Msg; omsg : Msg$
       init $ProxySocket = new(Socket(listenPort))$;
            $HTTPSocket = new(Socket(destinationPort))$
       do
            $\_incoming\_request\_ \neq \phi \rightarrow$
                $imsg := EncodeSOCK(DecodeHTTP(HTTPSocket.Read()))$;
                $\_incoming\_message\_ := ProxySocket.Write(imsg)$
            [] $\_outgoing\_request\_ \neq \phi \rightarrow$
                $omsg := EncodeHTTP(DecodeSOCK(ProxySocket.Read()))$;
                $\_outgoing\_message\_ := HTTPSocket.Write(omsg)$
       od
    ]|
```

address the uni-directional firewalls problem by adding a publicly addressable node, called "rendezvous server", that firewalled peer can already talk to. For the port-blocking firewalls, Microsoft has provided a similar solution via virtual tunnels with its Point-to-Point Tunneling Protocol (PPTP) [6]. This solution works satisfactorily with Windows machines, but this approach is based on the client-server networking model and unsuitable for the peer-to-peer networking model.

Previously [1], we have specified a Gnutella-like peer-to-peer system within the OO-action systems framework by combining UML diagrams. In this paper, we have presented our solution to traverse firewalls for peer-to-peer systems. We have extended a Gnutella-style peer-to-peer system to adopt uni-directional firewalls and port-blocking firewalls using OO-action systems. During the extending work, we continue our methodology that combines formal and informal methods together to extend the system. Besides the benefits from formal methods, our experiences also show that the object-oriented aspect of OO-action systems helps to build systems with a reuseable, composable and extendable architecture.

Peer-to-peer networking is currently attracting lots of attention, spurred by the surprisingly rapid deployment of some peer-to-peer applications like BitTorrent, Kazaa and eMule. Firewalls have become a great challenge to peer-to-peer networking upon Internet. In the future work, we plan to explore more sophisticated protocols like SOAP [7] and incorporate them into the development of peer-to-peer systems to provide safe and reliable access via firewalls.

Acknowledgements

The author is grateful to Kaisa Sere and Xinrong Zhou.

References

1. Lu Yan and Kaisa Sere: *Stepwise Development of Peer-to-Peer Systems.* Proceedings of the 6th International Workshop in Formal Methods (IWFM'03), Dublin, Ireland, July 2003. Electronic Workshops in Computing (eWiC), British Computer Society Press.
2. Clip2 DSS: *Gnutella Protocol Specification v0.4.* Online. http://www.clip2.com/GnutellaProtocol04.pdf.
3. E. Sekerinski and K. Sere: *A Theory of Prioritizing Composition.* The Computer Journal. Vol. 39, No.8, 1996.
4. L. Gong: *JXTA: A network programming environment.* IEEE Internet Computing, 5(3): 88–95, May/June 2001.
5. I. Ivkovic: *Improving Gnutella Protocol: Protocol Analysis and Research Proposals.* Technical report, LimeWire LLC, 2001.
6. MSDN Library: *Understanding Point-to-Point Tunneling Protocol (PPTP).* Online. http://msdn.microsoft.com.
7. W3C: *Simple Object Access Protocol (SOAP).* Online. http://www.w3c.org.

Paper III

SkyMin: A Massive Peer-to-Peer Storage System

L. Yan, M. F. Serra, G. Niu, X. Zhou, K. Sere

Available in: *Proceedings of the 3rd International Conference on Grid and Cooperative Computing (GCC'04), October 2004. Lecture Notes in Computer Science 3251, Springer-Verlag.*

SkyMin: A Massive Peer-to-Peer Storage System

Lu Yan[1], Moisés Ferrer Serra[2], Guangcheng Niu[2], Xinrong Zhou[1], and Kaisa Sere[1]

[1] Turku Centre for Computer Science (TUCS) and Department of Computer Science, Åbo Akademi University, FIN-20520 Turku, Finland
{lyan, xzhou, kaisa}@abo.fi
[2] Department of Computer Science, Åbo Akademi University, FIN-20520 Turku, Finland
{mferrer, gniu}@abo.fi

Abstract. The aim of the project *SkyMin* is to design a large-scale, Internet-based storage system providing scalability, high availability, persistence and security. Every node serves as an access point for clients. Nodes are not trusted; they may join the system at any time and may silently leave the system without warning. Yet, the system is able to provide strong assurance, efficient storage access, load balancing and scalability. Our approach to construct a massive storage file system on Internet is to implement a layer on top of existing heterogeneous file systems. The architecture is as follows: it consists of lots of FS (File Server) and one or some NS (Name Server). NS is the control center. Users can access the file system from every node.

1 Introduction

The aim of the project *SkyMin* is to build a global storage system, based on peer-to-peer (p2p) technology, and running on Internet [1] [2] [3] [4]. In that system we encounter a central node (Name Server) and a lot of peers (File Servers). The relationship between the peers and the name server is modeled as client-server paradigm, while between the peers themselves is pure p2p (they work as client and server at the same time). Every node can join the system at any time, and leave it silently (without warning).

The system is specially tailored for groups with limited budgets that need to store and share information in a secure way, making it easily available for everyone (the peers will store the information, and all the shared space will be available for the whole group). No special skills are required for the peer user, since the application values a user-friendly design; only some administrative skills are required for running the name server since the correct behavior of the system depends on the appropriate configuration and administration of the name server.

The remainder of the paper is organized as follows. We start with the requirement specification in Section 2. We present the architecture of *SkyMin* in Section 3 and more design details in Section 4. In Section 5, the implementation and test are discussed. We conclude the paper in Section 6.

2 Requirement Specification

As mentioned earlier, the system is based on the implementation of two different applications: the File Server (from now on just "peer") and the Name Server (from now on just "server"). While the server program will not be used "directly" by most end users (except the administrators), but the peer program will be.

2.1 Functionality

The functionality requirement of the system is intended to hide all the complexity of the system. Therefore, although more sophisticated functionality is provided (such as login/logout etc.), the main functionality of the system is the one described in Table 1.

Table 1. Main Functionality

Feature	Functionality that provide
Add User	Add users to a group (p2p network group)
Remove User	Remove an existing user from a group (p2p network group)

Server application functionality

Feature	Functionality that provide
Search	Allow the user to lookup files
Download	Get (read) files from the shared space
Upload	Put (write) files to the shared space
Delete	Delete a file from the shared space

Peer application functionality

2.2 More specification points

Although we have introduced the main functionality of the system from the user's point of view, we have to regard that the server side holds a very important role determining the communication between peers (each peer will be connected to the server, and it decides how the connections will be done). Therefore, we need to refine our functionality requirement to better specify the behavior of the system. We present a list with these important (and/or clarifying) points:

- Nodes can join and leave silently. There is no change in the data availability. Neither is there any change in the behavior of other nodes.
- Storing data means copying a file from a node (located in a user computer, but not in the shared space), to the storage space, if there is enough space in the storage system. If there is no space available, the system prompts an error message.
- The system must not store duplicated files in the same node.

– A name server may store information of several groups. A user should not be able to access information that is out of the scope of his/her own group.

– The system does not need to support a user running multiple instances of the program at the same time (for example, over different groups).

2.3 Non-functional requirements

Non-functional requirements are required for the system. Some of them are contradictory, so we have to make some tradeoff among these requirements:

– Scalability: there is no restriction on the number of nodes that can simultaneously join a group (and the performance must not be affected).

– High availability: redundancy in the storage system (allow duplicates and provide mechanisms/algorithms to guarantee the availability of resources).

– Persistence: the system should be capable of nonstop running.

– Security: a very important feature, since the information traveling over Internet might be private and should not be accessible to people out of the p2p group.

– Reliability: strongly related to persistence; the server side must be error tolerant. (Some kind of recovering mechanism must be provided for that.) The peer side should also be reliable (for example, the server crashes but the peer continues the communication with the other peers).

– Load balancing: store information in an efficient way, not randomly.

3 System Architecture

The *SkyMin* framework is a global storage system, as shown in Fig. 1, based on p2p technology. This system consists of one (or more) central server and many peers which are connected to Internet. The server works as a control point to grant the peer's access to the storage system, to manage the indexes of the peers' resources (files can be accessed by other peers who are members of the whole storage system), etc. On the other hand, every joined peer contributes a slice of their local storage space to the system, and the peer can save and retrieve files to/from the system.

With this picture in mind, it was clear that we were going to develop two different applications that should interact. Therefore we encapsulate each of them in a package: one for the peer and one for the server.

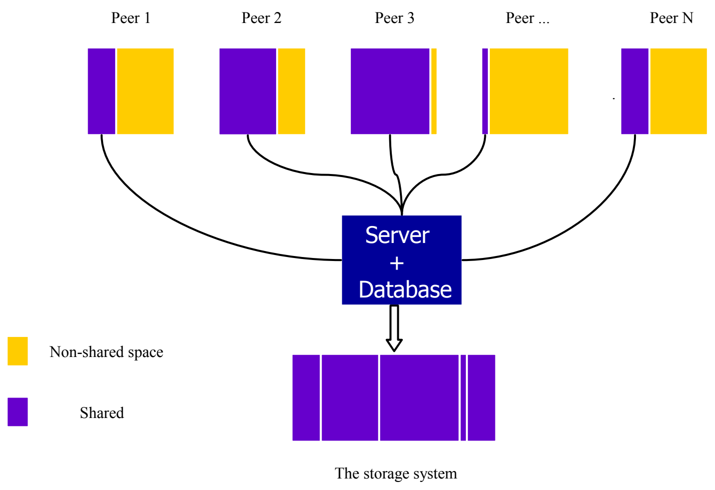


Fig. 1. System Architecture

3.1 Peer-side architecture

There are several tasks that the peer should be able to perform. These tasks can be summarized as:

- Receive and analyze the user's commands.
- Send its resource information to the server and maintain it periodically.
- Send user's requests to the server and process the server's response.
- Work as a server itself to respond to other peer's download or upload request.

Note that some of these tasks are more complicated than the other ones. Therefore several classes are used for them. According to the requirement specification of our peer system, we built the following classes to accomplish these tasks in Fig. 2:

The "Peer" class holds the main logic of the peer program in the peer application. It starts the system, (taking care of program variables, server in the background, login, etc) and accepts commands from the user and process them (doing corresponding actions). The "Shell" class handles the user input; it parses the input. If the user's command makes no sense, this class will prompt an error message. The "ProtocolMsg" class holds the information of building different messages for different purposes (e.g. search, upload, etc.). This class keeps the protocol information of within messages (each message has a unique identifier, and this class keeps that information). The "PeerResource" class is used for constructing a message for sending the peer resources. The class allows sending different kinds of resources, including peer space (maximum available space) and files.

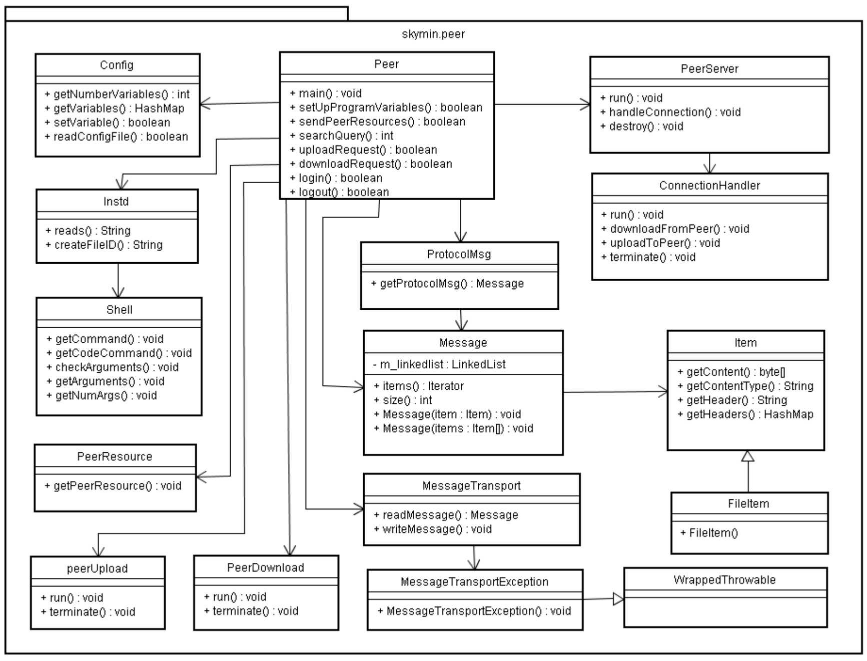

skymin.peer
Config
+ getNumberVariables() : int
+ getVariables() : HashMap
+ setVariable() : boolean
+ readConfigFile() : boolean
Peer
+ main() : void
+ setUpProgramVariables() : boolean
+ sendPeerResources() : boolean
+ searchQuery() : int
+ uploadRequest() : boolean
+ downloadRequest() : boolean
+ login() : boolean
+ logout() : boolean
PeerServer
+ run() : void
+ handleConnection() : void
+ destroy() : void
ConnectionHandler
+ run() : void
+ downloadFromPeer() : void
+ uploadToPeer() : void
+ terminate() : void
Instd
+ reads() : String
+ createFileID() : String
ProtocolMsg
+ getProtocolMsg() : Message
Shell
+ getCommand() : void
+ getCodeCommand() : void
+ checkArguments() : void
+ getArguments() : void
+ getNumArgs() : void
Message
- m_linkedlist : LinkedList
+ items() : Iterator
+ size() : int
+ Message(item : Item) : void
+ Message(items : Item[]) : void
Item
+ getContent() : byte[]
+ getContentType() : String
+ getHeader() : String
+ getHeaders() : HashMap
PeerResource
+ getPeerResource() : void
MessageTransport
+ readMessage() : Message
+ writeMessage() : void
FileItem
+ FileItem()
peerUpload
+ run() : void
+ terminate() : void
PeerDownload
+ run() : void
+ terminate() : void
MessageTransportException
+ MessageTransportException() : void
WrappedThrowable

Fig. 2. Peer Class Diagram

It is the “MessageTransport” class that takes the responsibility of the message transportation over Internet. Possible exceptions that may occur during this transportation are built on the “MessageTransportException” class which extends the “WrappedThrowable” class. So far, all messages are sent in a homogeneous way, thanks to the protocol. Messages are sent identically; with the identifier provided with the “ProtocolMsg” class, they can be properly interpreted.

The peer also runs a server thread in the background, which is an object of the “PeerServer” class. This class implements a listener to a specific port; when a message arrives (a peer wants to connect), the process will be controlled by a “ConnectionHandler” object.

The “Item” class is used to build the content of messages. Items are text ones, but can be extended to support files. This is done by the class “FileItem”. Items are finally encapsulated in a message (the “Message” class) and therefore can be sent. The “Instd” class provides several functions with different purposes. It is like a utility class which builds the identifier of files and also takes care of reading from the standard input.

The “PeerDownload” class is responsible to manage the process when the peer wants to download a file. It connects the peer and updates the pertinent information in the server. The “PeerUpload” class has the same purpose as the previous class except that the peer wants to perform an upload. The “Config” class is responsible for reading the configuration file and load program variables.

3.2 Server-side architecture

There are also several tasks that the server should be able to perform. These can be summarized as:

- Receive and analyze the user's commands.
- Listen to a port for incoming connections.
- Handle incoming connections.
- Interpret peer messages and send proper answers.
- Add or remove users to/from a group.

As before, some functions are more complex than the other ones. Therefore they will be implemented in several classes. The server class diagram is shown in Fig. 3.

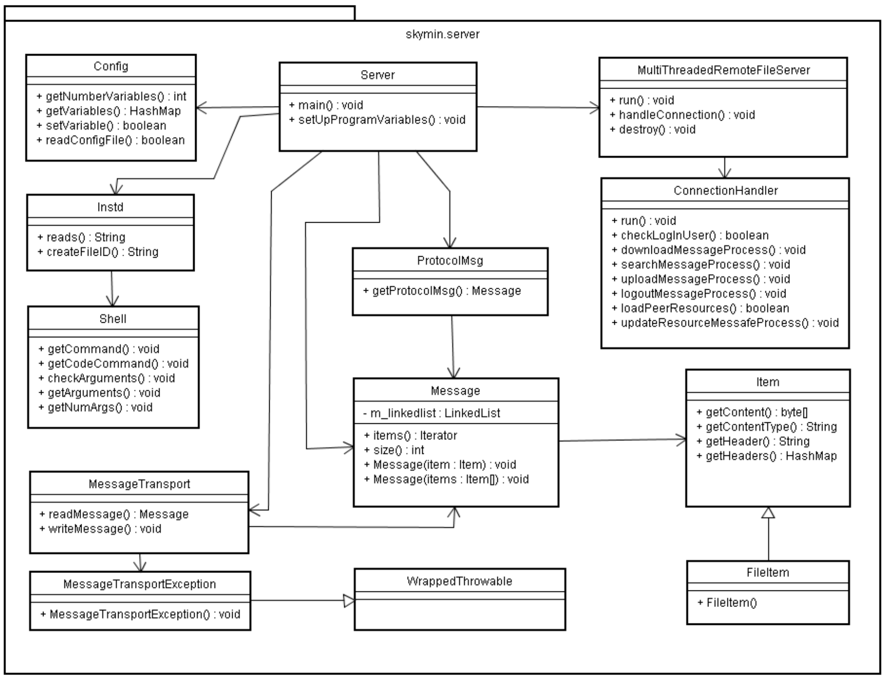

skymin.server
Config
+ getNumberVariables() : int
+ getVariables() : HashMap
+ setVariable() : boolean
+ readConfigFile() : boolean
Server
+ main() : void
+ setUpProgramVariables() : void
MultiThreadedRemoteFileServer
+ run() : void
+ handleConnection() : void
+ destroy() : void
Instd
+ reads() : String
+ createFileID() : String
ConnectionHandler
+ run() : void
+ checkLogInUser() : boolean
+ downloadMessageProcess() : void
+ searchMessageProcess() : void
+ uploadMessageProcess() : void
+ logoutMessageProcess() : void
+ loadPeerResources() : boolean
+ updateResourceMessafeProcess() : void
ProtocolMsg
+ getProtocolMsg() : Message
Shell
+ getCommand() : void
+ getCodeCommand() : void
+ checkArguments() : void
+ getArguments() : void
+ getNumArgs() : void
Message
- m_linkedlist : LinkedList
+ items() : Iterator
+ size() : int
+ Message(item : Item) : void
+ Message(items : Item[]) : void
Item
+ getContent() : byte[]
+ getContentType() : String
+ getHeader() : String
+ getHeaders() : HashMap
MessageTransport
+ readMessage() : Message
+ writeMessage() : void
MessageTransportException
+ MessageTransportException() : void
WrappedThrowable
FileItem
+ FileItem()

Fig. 3. Server Class Diagram

Some classes have the same behavior here as in the peer side, even when working in a little different way (one clear example is the shell, which carries out the same work but process different commands). Therefore "Shell", "Config", "Instd", "Item", "FileItem", "Message", "MessageTransport", "MessageTransportException", "WrappedThrowable" and "ProtocolMsg" do not need further explanation.

The "Server" class is like the "Peer" class in the peer side. It performs all the initializations and prepares the system for working. The "MultithreadedRemoteFile-Server" is analogous to the "PeerServer" in the other package. It implements the listener to a specific port for incoming connections. The “ConnectionHandler” is used to handle the incoming connections.

4 System Design

The behavior of the system can be explained by sequence diagrams. We will see sequence diagrams for the peer and for the server.

In the sequence diagrams there are some descriptions like *Msg(keyword)*. This defines the protocol of our system; it means that the system expects some messages and reacts to the received one and/or the expected one. That is the reason between peer and server and between peer and peer all the descriptions are of the type *Msg(x)*. These keywords are implemented in the class “ProtocolMsg”.

4.1 Peer-side dynamic design

Login. The first important scenario that we have (not use case) is the login process in Fig. 4.

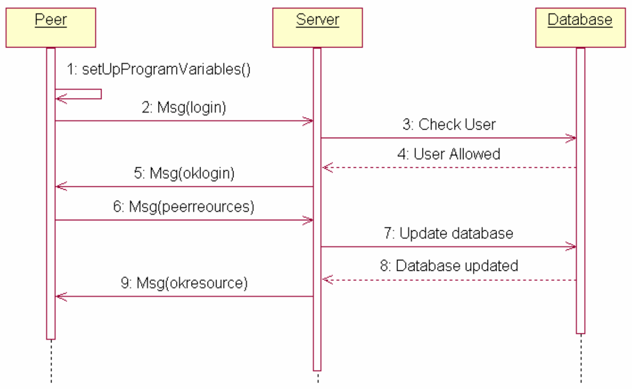


Fig. 4. Login sequence diagram

In the login process we can see that first of all, the peer loads the variables, then it tries to connect the server, providing user name and password, encapsulated in the login message. The server checks the database whether the user is allowed and notifies the peer. Then the peer sends the information of its resources to the server, encapsulated in a *perresources* message. The server tries to update the databases with this information, and if the update is performed successfully, the server notifies it to the peer with an *okresources* message. If any operation fails, the peer will abort starting the program.

Search. Here the situation is easy to understand in Fig. 5. The peer sends a *search* message that contains a query, in order to find the file. The server passes the query to the database and retrieves the result. If the result is not empty (the normal case), the server will answer the peer with an *oksearch* message. Otherwise, the server answers with an *emptyquery* message.

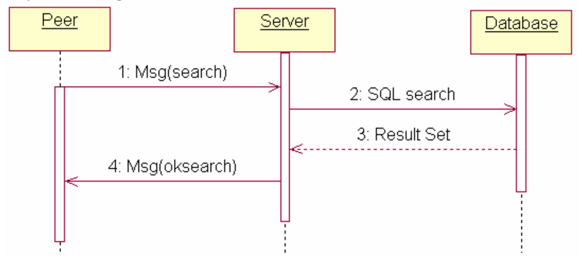


Fig. 5. Search sequence diagram

Download. In the download scenario in Fig. 6, first of all a search is needed (before the user downloads a specific file, he/she needs to know where it is). The following steps 1 to 4 are the same as a normal search. After that, the user selects one of the files provided by the search operation to download, and sends the information to the server in a *downloadrequest* message. The server searches the database for the best available peer, and answers with an *okdownloadrequest* message. Then the peer connects the other peer asking for the file (*download* message), and the second peer answers as well (*startdownload* message). Finally, when the download completes, the peer will update its information resources in the database.

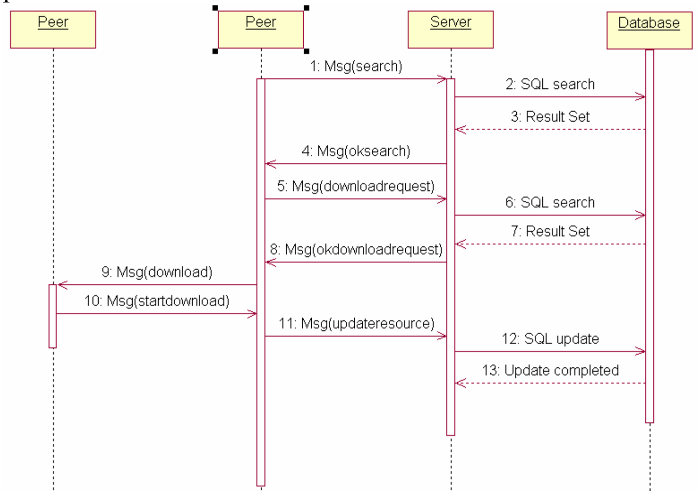


Fig. 6. Download sequence diagram

Different variations can be appreciated in the system. For example, if there is no suitable peer for download, the server will answer with an *emptyquery*. So the peer realizes it. Also an *error* message can come from the second peer, aborting the downloading process.

Upload. Here the situation is shown in Fig. 7. The peer sends an *uploadrequest* message, telling the ID and the size of the file to upload. Then the server checks the database which peer is the best to upload, and answers an *okuploadrequest* (note that the server can also answer an *uploadnotrequired* message), specifying the best peer to upload (note that the own peer that makes the request can also be the peer chosen). Then the peer contacts the chosen peer by an *upload* message, and waits for the answer (*startupload*). Finally, after the file is sent, the peer informs the server about its new resources, and the server updates the database.

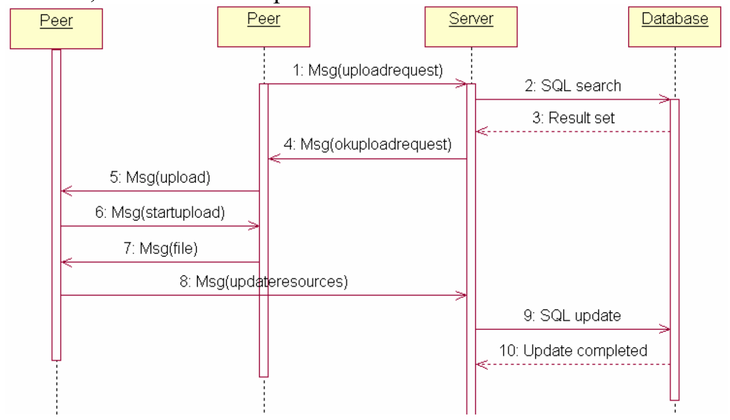


Fig. 7. Upload sequence diagram

Delete. The diagram in Fig. 8 is quite simple. The user deletes the file from his/her own shared space with the delete command. Then the peer sends an *updateresource* message to the server, who updates the database.

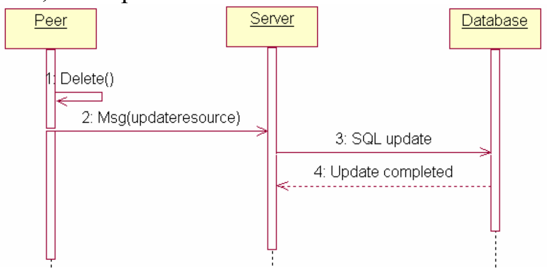


Fig. 8. Delete sequence diagram

Exit. As shown in Fig. 9, when leaving the program, the peer sends a message to the server (*logout*) to notify it. The server deletes all the references in the tables of this peer, and then sends an *oklogout* message to the peer.

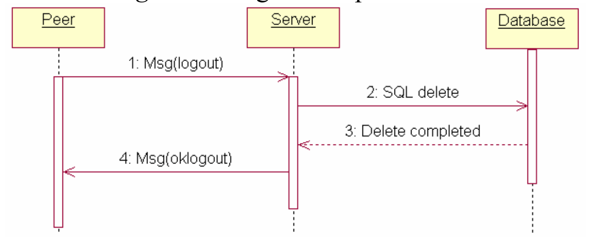


Fig. 9. Logout sequence diagram

4.2 Server-side dynamic design

Add user. When adding a user to the group, the command is received from the standard input and processed in the shell class. After the system parses it, an SQL grant statement will be executed, and the database will be updated. The diagram is shown in Fig. 10.

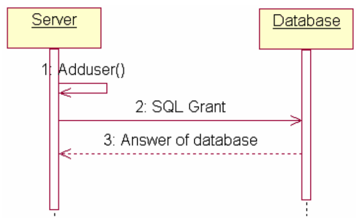


Fig. 10. Add user sequence diagram

Remove user. Similar to the add user command, except that the SQL statement to be executed is a revoke instead of a grant. The diagram is shown in Fig. 11.

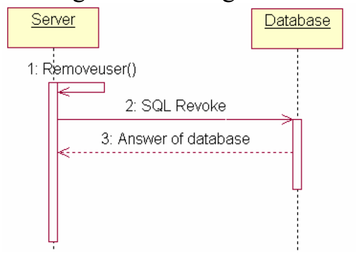


Fig. 11. Remove user sequence diagram

Exit. When the server makes an exit, first it needs to destroy (close) the thread that is listening to the port for incoming connections. After that it clears all the tables in *skymin* database, so that the tables will be consistent when the server restarts again (no rows in any table). The diagram is shown in Fig. 12.

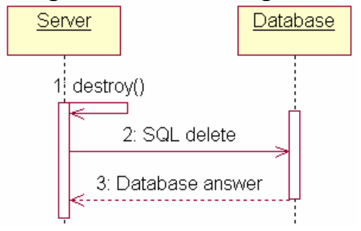


Fig. 12. Exit sequence diagram

5 Implementation and Test

The *SkyMin* application is implemented in Java 1.4.2, a language that is platform independent. Therefore, it can be run in any platform with JVM support; but since third party programs are used, in order to run it properly, some requirements are needed in the system:

– Java Runtime Environment (JRE v1.4 or higher)

– Connection to Internet (owning a valid IP address)

– MySQL (v4.0.18 or higher)

– ODBC driver for MySQL.

5.1 Database

The *SkyMin* uses only one database, named *skymin* and managed by MySQL. The peer application inserts and retrieves information to/from the database, but through the server application. So it is the server side that takes the responsibility of accessing the database. Our database is compound by two tables:

– Users: this table keeps track of the information of every peer that joins the network currently (maintains the information of the users that are logged in).

– Files: this table stores the information of the files that every user has (one entry for each file a user has).

Users table. The users table has 7 fields:

- UserID: this stores the IP of the user, which is used as its identification. Therefore it is the primary key of the table. This field is a variable string of 30 or less characters.
- UserName: here we store the alias of the user (the one he/she typed when logging in). This field is a variable string of 16 or less characters.
- Port: it stores the port number that the peer is using for listening incoming requests. It is an integer.
- MaxSpace: this field stores the maximum capacity of the sharing folder in bytes.
- FreeSpace: this field is similar to the previous one. It keeps information about how much free space (also in bytes) currently in the peer.
- Ratio: this field is a float result of the division *FreeSpace/MaxSpace*. It is used in some algorithms in our program.
- Counter: this field is an integer used to control the state of the peers (alive or not). This field is reserved for the future, not used in the current release.

Files table. The files table has 4 fields:

- FileID: is a unique identifier of the file. It is a variable char of 32 bytes.
- FileName: the name of the file. It has a limit of 255 characters.
- FileSize: the size of the file.
- UserID: this stores the owner of the file (where the file is located). It is used as a relating key with the other table.

Queries. The database answers to the following queries in different scenarios:

- Insert a row to the users table: when a user logs in.
- Delete a row from the users table: when a user logs out.
- Modify the users table: when the program modifies the content of the shared space of a peer, *FreeSpace* and *Ratio* must be properly updated.
- Insert a row to the files table: when a new file appears in the shared space.
- Delete a row from the files table: when a file disappears from the shared space.
- Search the files table by name: when a peer makes a search request.
- Search the files table by FileID: when a peer makes a download or upload request.
- Search the users table by Ratio: when a peer makes an upload request.
- Count users and files: when a peer makes an upload request.

5.2 Test

Since Java is our main programming language, we use JUnit for testing it. The approach of compiling overnight without error and test by hand is also used. In this way, we minimize the amount of errors in further testing phases. A systematic summary of the testing methodologies and time lines that we used to test our system is shown in Table 2.

Table 2. Task-oriented functional tests

	Module	Integrated	System	Stress	Regression	Acceptance
Timeline	After finishing each module	After module test	After integrated test	After system test	After modification, before an alpha, beta, final release	After regression test, when delivering a stable release
Iter. 1	OK	OK	OK	OK	OK	Not needed
Iter. 2	OK	OK	OK	OK	OK	Not needed
Iter. 3	OK	OK	OK	OK*	OK*	OK

The element OK* reflects the fact that despite that no more errors have been detected, the test team agrees on the need of more tests in order to assure a high quality product, free of errors.

6 Concluding Remarks

The increasing demand for massive storage systems has spawned an urge for a large-scale storage solution with scalability, high availability, persistence and security. Nowadays, Internet has become cheap and widespread, which makes it possible to build an economical massive storage solution over Internet.

In this paper, we proposed *SkyMin*, a global storage system, based on peer-to-peer (p2p) technology, and running on Internet, specially tailored for groups with limited budgets to store and share information in a secure way. This paper presented a prototype implementation of the architecture and design elements of *SkyMin*; several design elements still need fine-tuning (e.g. in the current release, there is no support for transport-level security; more sophisticated protocols like SSH will be incorporated in the next release). The rise of pervasive computing has brought new innovative design ideas for our architecture. In the future work, more design elements will be further refined in favor of ubiquity and mobility to make *SkyMin* a massive storage solution for the pervasive computing era.

Acknowledgements

We would like to acknowledge the support from Elena Troubitsyna.

References

1. L. Yan and K. Sere: Developing Peer-to-Peer Systems with Formal and Informal Methods. In *Proceedings of the 2nd International Workshop on Refinement of Critical Systems: Methods, Tools and Developments (RCS'03)*. Turku, Finland, June 2003.
2. L. Yan and K. Sere: Stepwise Development of Peer-to-Peer Systems. In *Proceedings of the 6th International Workshop in Formal Methods (IWFM'03)*. Dublin, Ireland, July 2003. Electronic Workshops in Computing (eWiC), British Computer Society Press.
3. L. Yan, K. Sere, and X. Zhou: Peer-to-Peer Networking with Firewalls. In *WSEAS Transactions on Computers*, 3(2): 566-571, 2003.
4. L. Yan, K. Sere, X. Zhou, and J. Pang: Towards an Integrated Architecture for Peer-to-Peer and Ad Hoc Overlay Network Applications. In *Proceedings of the 10th IEEE International Workshop on Future Trends in Distributed Computing Systems (FTDCS 2004)*. Suzhou, China, May 2004. IEEE Computer Society Press.

Paper IV

Building a Formal Framework for Mobile Ad Hoc Computing

L. Yan, J. Ni

Available in: *Proceedings of the International Conference on Computational Science (ICCS'04), June 2004. Lecture Notes in Computer Science 3036, Springer-Verlag.*

Building a Formal Framework for Mobile Ad Hoc Computing

Lu Yan and Jincheng Ni

Turku Centre for Computer Science (TUCS) and
Department of Computer Science, Åbo Akademi University,
FIN-20520 Turku, Finland.
{Lu.Yan, Jincheng.Ni}@abo.fi

Abstract. Mobile Ad Hoc Network (MANET) is an autonomous system of mobile nodes connected by wireless links. MANET does not require any fixed infrastructure such as base stations, therefore, it is an attractive networking option for connecting mobile devices quickly and spontaneously. This paper presents a formal framework towards a systematic design for MANET applications. In this paper, we define a layered architecture for mobile ad hoc computing and propose a middleware layer with three key components between software application layer and ad hoc networking layer. A formal specification for mobile ad hoc computing in this architecture is derived in the B method.

1 Introduction

A mobile ad hoc network (MANET) is a self-organizing and rapidly deployable network in which neither a wired backbone nor a centralized control exists. The network nodes communicate with one another over wireless medium. Nodes can only directly communicate with their neighbors. Therefore, distant nodes must communicate with one another in a multi-hop fashion [1]. MANET is adaptable to highly dynamic topology resulted from mobility of network nodes and does not require any fixed infrastructure such as base stations, therefore, it is an attractive networking option for connecting mobile devices quickly and spontaneously [12].

To develop such a system for mobile ad hoc computing, we define a layered architecture in Fig. 1 with three key components: *Network Management, Awareness* and *Interaction*, which buildup a middleware layer between software application layer and ad hoc networking layer. We specify the system components with the B method [6][7][8], and model the interactions and message communications between components with UML diagrams [11].

In this paper, we present a formal framework for mobile ad hoc computing in the *Ad Hoc Computing* project at TUCS that we are currently working on, and a complete detailed specification of mobile ad hoc computing and some experiment results can be found at [9].

The rest of the paper is structured as follows. Section 2 defines some pre-definitions of MANET context and environment. The specification of three key components in the architecture is presented in Section 3. We end in Section 4 with concluding remarks.

2 MANET Context and Environment

In order to specify the MANET context and environment, we define MACHINE *AdHocNet* and *RouteInfo* in the B language, which are to be referred in the specification of system components in the next section. *AdHocNet* is defined for the general context and environment of MANET and *RouteInfo* is defined as the routing information in mobile ad hoc computing. As an example, *AdHocNet* is specified as follows.

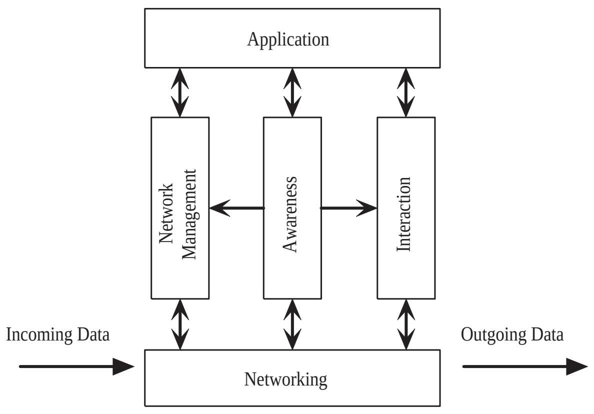


Fig. 1. Layered Architecture

MACHINE *AdHocNet*

SETS

NODES **;** *CommMSG* = { *commMsg* , *routeError* } **;**
RouteMSG = { *routeReq* , *routeRep* }

CONSTANTS

myID

PROPERTIES

$myID \in NODES$

END

In the *AdHocNet* specification, there are three sets: *NODES, CommMSG* and *RouteMSG*. They are defined as set of nodes, set of communication messages, and set of routing messages in ad hoc networks. The *CommMSG* has two different elements: *commMsg* and *routeError*. The *commMsg* is used for the communication between nodes. The *routeError* is used when a route is broken. The *RouteMSG* consists of two kinds of routing messages: *routeReq*(route request) and *routeRep*(route reply), which are used in the *Awareness* component to detect remote nodes. The node's ID *myID* is a constant in the *AdHocNet*, which is an important property of NODES to identify a node in networks.

3 MANET Architecture

3.1 Network Management

There is no constant topology or centralized manager in MANET. In order to form a self-organizing network, and support multi-hop routing by forwarding packets, it is necessary to have the network management in every node in MANET.

In the specification of network management, there are three B machines: *netManager, modeSet* and *Connector*. As shown in Fig. 2, *netManager* includes *modeSet* and *Connector* and calls their operations.

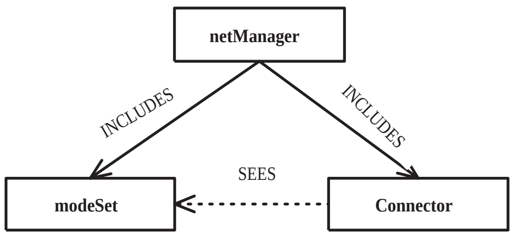


Fig. 2. Network Management

There are two modes to be set in *modeSet*: discoverable or non-discoverable. When a node is in discoverable mode, it can be discovered by other nodes. Otherwise, it cannot be discovered in MANET. The *Connector* is used to connect or disconnect neighbor nodes in MANET. With these machines, a node can form, join or leave networks via connecting or disconnecting neighbor nodes when it is in discoverable mode.

The *netManager* manages the activities of the system and update connection status of neighbor nodes. When shutting down the system, the *netManager* disconnects all connections and set the node into non-discoverable mode. It has the following operations:

- *joinNet* - join the network and connect all detected nodes (used at the network setup stage).
- *updateNet* - connect local nodes actively when the host has detected some unconnected nodes (used to update the network topology).
- *leaveNet* - set mode into non-discoverable and disconnect all linked nodes.

```
MACHINE    netManager
SEES
   awareNodes , AdHocNet
INCLUDES
   modeSet , Connector
OPERATIONS
   joinNet  ≙
     PRE    $nodeMode = discoverable \wedge LOCALNODES \neq \{\}$
     THEN   connectAll ( LOCALNODES )
     END  ;
   updateNet  ≙
     PRE    $nodeMode = discoverable \wedge LOCALNODES - LINKND \neq \{\}$
     THEN
        IF     $LOCALNODES = \{\}$
        THEN   disconnectAll
        ELSIF  $LINKND = \{\}$
        THEN   connectAll ( LOCALNODES )
        ELSE
           ANY    local_nd , linked_nd
```

```
            WHERE   $local\_nd \in LOCALNODES \wedge$
               $linked\_nd \in LINKND$
            THEN
               SELECT   $local\_nd \notin LINKND$
               THEN   $connect ( local\_nd )$
               WHEN   $linked\_nd \notin LOCALNODES$
               THEN   $disconnect ( linked\_nd )$
               END
            END
         END
      END   ;

   leaveNet   $\widehat{=}$
      PRE   $true$
      THEN   $setNDiscovered \parallel disconnectAll$
      END

END
```

3.2 Awareness

Awareness of mobile computing is used to sense a certain environment in order to present or update context of mobile systems [2]. In this paper, we focus on detecting node ID and processing incoming date packets.

Node Detections There are two kinds of node detections: *local awareness* - system can detect a local node within the radio transmission range; *remote awareness* - system should also be able to detect a friendly remote node with a known ID. In local awareness, a node detects its neighbor nodes and the detected nodes will be connected and used to update the topology in *Network Management*. In remote awareness, a node tries to find out routes to connect a friendly remote node with a known ID.

Incoming Message Processing In message awareness, a node processes incoming messages according to the format of data packets. There are two kinds of messages in MANET: communication message and routing message. If the received message is a communication message, the system checks the packet head, and then receives or forwards the packet according to the next hop ID of the route. In case the ID is unrecognizable, the system will report a broken route. There are two kinds of routing messages: route request and route reply. The route request *routeReq* is a message for detecting a remote node and the route reply *routeRep* is a reply message when the destination node receives a *routeReq*. If the incoming message is a routing message, the system will process the message according to the current routing protocols in MANET.

```
MACHINE   $awareMSG$

SEES

   $modeSet$ , $AdHocNet$ , $Connector$

INCLUDES
```

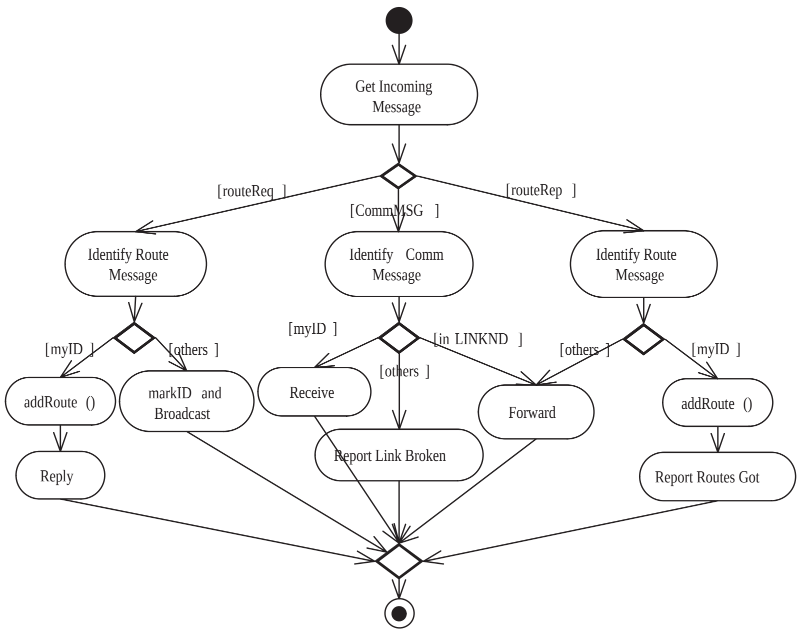


Fig. 3. Incoming Message Processing

```
    $RouteInfo$
SETS
    $ACK = \{\ receive\ ,\ forward\ ,\ get\_routes\ ,\ link\_break\ ,\ route\_reply\ ,\ markID\_broadcast\ \}$
VARIABLES
    $identifyCommMsg\ ,\ identifyRouteMsg\ ,\ getRouteInfo$
INVARIANT
    $identifyCommMsg \in CommMSG \rightarrowtail NODES\ \wedge$
    $identifyRouteMsg \in RouteMSG \rightarrowtail NODES\ \wedge$
    $getRouteInfo \in RouteMSG \twoheadrightarrow ROUTES$
INITIALISATION
    $identifyCommMsg := \{\}\ \|\ identifyRouteMsg := \{\}\ \|$
    $getRouteInfo := \{\}$

OPERATIONS
    $ans \longleftarrow$ awaringMSG $\widehat{=}$
      PRE  $nodeMode = discoverable$
      THEN
        CHOICE  ANY  $msg$
          WHERE  $msg \in CommMSG$
          THEN
            IF  $identifyCommMsg\ (\ msg\ ) = myID$
            THEN  $ans := receive$
          ELSIF   $identifyCommMsg\ (\ msg\ ) \in LINKND$
          THEN    $ans := forward$
          ELSE    $ans := link\_break$
          END
       END
    OR   ANY   $msg$
       WHERE   $msg \in RouteMSG$
       THEN
          SELECT   $msg = routeReq$
          THEN
             IF   $identifyRouteMsg\ (\ msg\ ) = myID$
             THEN
                $addRoute\ (\ getRouteInfo\ (\ msg\ )\ )\ \ ||$
                $ans := route\_reply$
             ELSE   $ans := markID\_broadcast$
             END
          WHEN   $msg = routeRep$
          THEN
             IF   $identifyRouteMsg\ (\ msg\ ) = myID$
             THEN
                $addRoute\ (\ getRouteInfo\ (\ msg\ )\ )\ \ ||$
                $ans := get\_routes$
             ELSE   $ans := forward$
             END
          END
       END
    END
 END
END
```

In MACHINE *awareMSG*, there are three important functions: *identifyCommMsg* is a function to identify the ID to whom the communication message is sent; *identifyRouteMsg* is a function to check to whom the routing message is sent and *getRouteInfo* is a function to get route information from *RouteMSG*. The operation *awaringMSG* checks incoming messages and processes them. The processing procedure is shown in the above specification and the activity diagram of message processing is shown in Fig. 3.

3.3 Interaction

Interaction mainly concerns communication links between nodes. We consider an opening session for the interactive communication between nodes. In such a session, the source and destination nodes exchange messages and update routing information for communication.

As shown in Fig. 4, when the system opens such a session and starts interactive communication, the source node will select a route from the routing table or detect a new route to reach the destination node. If there is no available route or the destination node is not detected in the network, the opening session fails and a failure message is sent back to the source node. In a

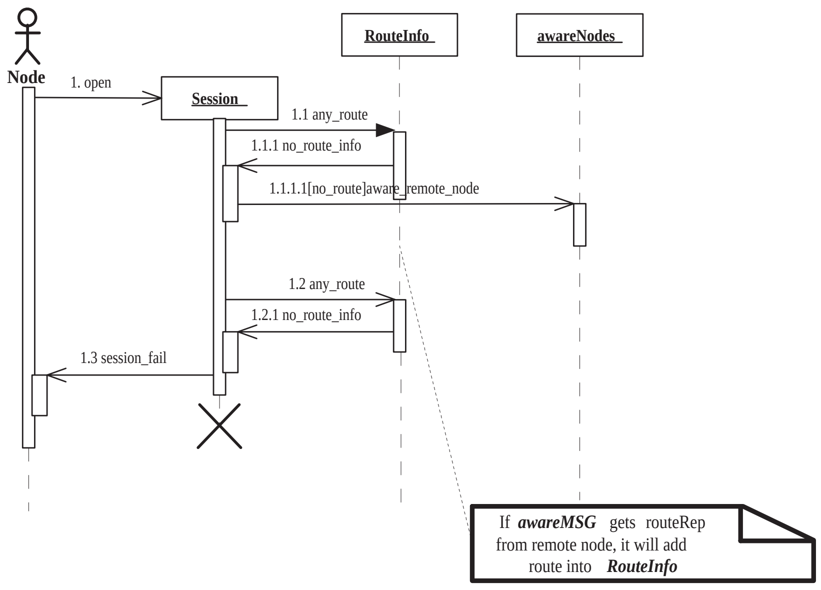


Fig. 4. Opening Session for Interactive Communication

successful case, once a route is available, a communication session between the source node and destination node is created and the interactive communication starts.

There are two parts in the specification of *interaction*: procedure of interactive communication and route maintenance when a broken route is detected.

Sending Messages and Communication Before sending a message, a node should select a route from the routing table. Once a route is selected, the node can start communication and messages are sent via the route. If the node cannot select any route, the system will notify that no route is available. It may happen due to a broken route in the interactive session, and hereby route maintenance and recovery are needed. In a successful case, the source and destination nodes are in interactive communication and incoming messages are checked and processed. The specification is shown as follows.

MACHINE *Communication*

SEES

modeSet , *AdHocNet*

INCLUDES

RouteInfo

SETS

$STATE = \{ \, INTERACT \, , NOROUTE \, , BREAK \, \}$

VARIABLES

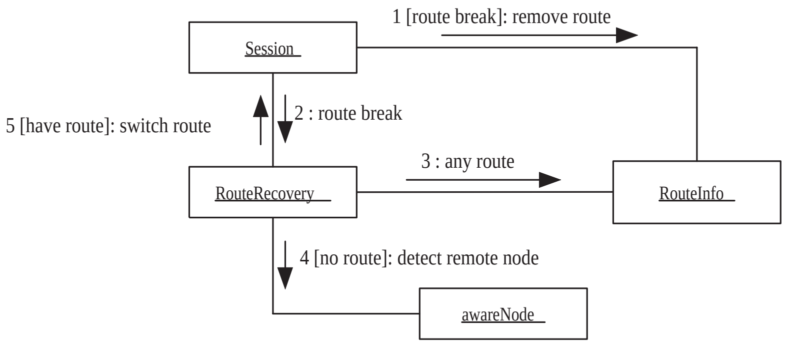
SendCommMSG , *currentState* , *commRoute*

INVARIANT

```
    $SendCommMSG \in CommMSG \rightarrowtail \mathsf{ran}\ (\ routeInfo\ ) \wedge$
    $currentState \in STATE \wedge$
    $commRoute \in \mathsf{ran}\ (\ routeInfo\ )$
INITIALISATION
    $SendCommMSG := \{\}\ \|\ currentState := BREAK\ \|\ commRoute := \{\}$

OPERATIONS
    get_commRoute ( $nd$ ) $\widehat{=}$
      PRE   $nd \in NODES$
      THEN
        ANY   $selectedRoute$
        WHERE   $selectedRoute \in \mathsf{ran}\ (\ routeInfo\ )$
        THEN
          IF   $nd \in selectedRoute$
          THEN   $commRoute := selectedRoute$
          ELSE   $currentState := NOROUTE$
          END
        END
      END   ;
    comm ( $msg$ ) $\widehat{=}$
      PRE   $msg \in CommMSG \wedge commRoute \neq \{\}$
      THEN   $SendCommMSG\ (\ msg\ ) := commRoute\ \|$
        $currentState := INTERACT$
      END   ;
    switchRoute ( $route$ ) $\widehat{=}$
      PRE   $route \in \mathsf{ran}\ (\ routeInfo\ )$
      THEN   $commRoute := route$
      END   ;
    routeBreak $\widehat{=}$
      PRE   $currentState = INTERACT \wedge commRoute \neq \{\}$
      THEN
        ANY   $msg$
        WHERE   $msg \in CommMSG \wedge msg = routeError$
        THEN
          $removeRoute\ (\ routeInfo^{-1}\ (\ commRoute\ )\ )\ \|$
          $currentState := BREAK$
        END
      END
END
```

Route Maintenance and Recovery During the interactive communication, the network topology might be changed and it might lead to a broken route. Thus route maintenance and recovery are needed for interactive communication. Figure 5 shows how a route is recovered when the system knows that the route is broken. In our design, it is assumed that multiple routes discovery protocols are used. For example, when source node S is communicating with destination node D, S sends data packets to D along with the selected route. During their communication, if S gets to know that the communication route is broken, S doesn't need to rediscover a new route immediately because S might have detected several routes in the previous discovery. It can then choose another available route and replace the broken one. Until all the routes are not reachable to the destination, the system will start route discovery again [10].



1 [route break]: remove route
Session
2 : route break
5 [have route]: switch route
3 : any route
RouteRecovery
RouteInfo
4 [no route]: detect remote node
awareNode

Fig. 5. Route Maintenance and Recovery

3.4 Relationship of Components

There are three components in the system specification, which are built up with nine B machines. Two pre-defined machines *AdHocNet* and *RouteInfo* are used to specify the context and environment of mobile ad hoc computing. The component *Network Management* is composed of three machines: *netManager, modeSet* and *Connector*, and the component *Awareness* has two machines: *awareNodes* and *awareMSG*. There are two machines: *Communication* and *RouteRecovery* in the component *Interaction*. For the whole system, the relationship of machines within components and between components is shown in Fig. 6.

4 Concluding Remarks

This paper specifies a formal framework towards the application development for ad hoc computing. In our system, three key components model different aspects of mobile ad hoc computing. Due to the mobility of nodes in MANET, the system is focused on the awareness of its environments, the connections between nodes and the communication of nodes. The routing problems are also considered when nodes are communicating with each other.

The goal of the specification is a formal framework to enable applications to be developed based on the three components, which are to be executed arbitrarily in MANET. In our current implementation of this framework via a Bluetooth platform [3], two main experiments have been done towards developing and testing mobile computing nodes in MANET. The first one is a peer-to-peer chatting application *BlueChat* running on Bluetooth devices [4]. The second one is an experiment of ad hoc network establishment and routing testing using LAN Access Profile of Bluetooth wireless technologies [5].

As a future work, we expect to improve the system to support multi-routing protocols. In the awareness computing aspect of MANET, the system will be designed to be more intelligent, i.e.

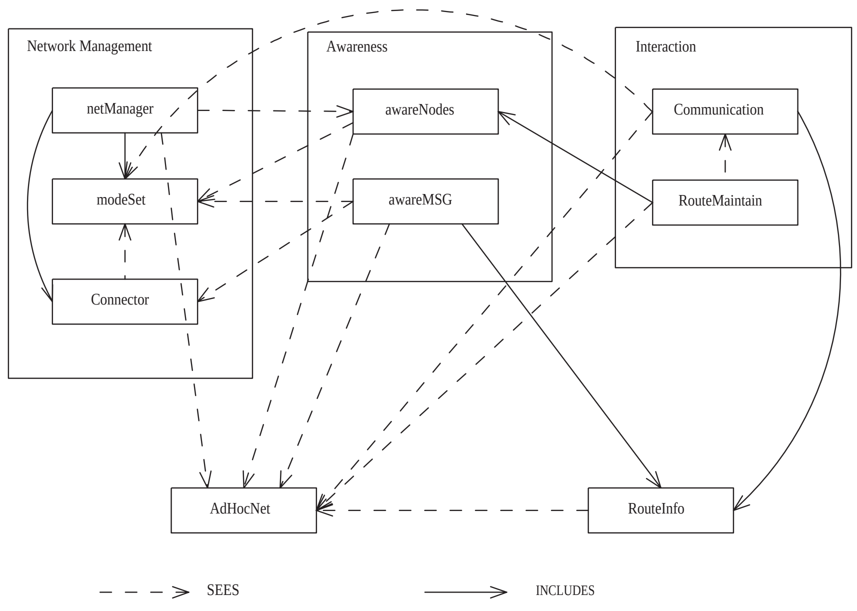


Fig. 6. Relationship of Components

service awareness, task awareness will be incorporated into the *Awareness* component. Ultimately, we aim at developing such a system as a middleware to support different platforms to enable applications to run compatibly on different devices.

Acknowledgements

This work was supported by the *Mobile Ad Hoc Computing* project directed by Kaisa Sere, and we are grateful for Kaisa's contribution.

References

1. E. M. Royer and C.-K.Toh, *A Review of Current Routing Protocol for Ad hoc Mobile Wireless Networks*, IEEE Personal Communications 6(2), 46-55, April 1999.
2. T. Selker and W. Burleson, *Context-aware Design and Interaction in Computer System*, IBM Systems Journal, Vol 39, No 3 & 4, 2000.
3. The Offical Bluetooth Website, http://www.bluetooth.com.
4. Bluetooth SIG, *Bluetooth Specification*, http://www.bluetooth.com.
5. C. M. Cordeiro, S. Abhyankar, R. Toshiwal and D. P. Agrawal, *A Novel Architecture and Coexistence Method to Provide Global Access to/from Bluetooth WPANs by IEEE 802.11 WLANs*, IEEE IPCCC, Phoenix, Arizona, April 2003.
6. J. R. Abrial, *The B-Book: Assigning Programs to Meaning*, Cambridge University Press, 1996.
7. K. Lano and H. Haughton, *Specification in B: An Introduction Using the B Toolkit*, Imperial College Press, London, 1996.
8. E. Sekerinski and K. Sere (Eds), *Program Development by Refinement: Case Studies Using the B Method*, Springer-Verlag, 1999.
9. J. Ni, *Towards a Systematic Design for Ad hoc Network Applications*, Master Thesis, Åbo Akademi, 2003.
10. Z. Ye, S. V. Krishnamurthy and S. K. Tripathi, *A Framework for Reliable Routing in Mobile Ad Hoc Networks*, Proceedings of the IEEE INFOCOM 2003, San Francisco, USA, 2003.
11. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns*, Addison-Wesley, 1995.

12. S. Corson and J. Macker, *Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations*, Tech. Rep. RFC 2501, IETF, Jan. 1999.

Paper V

Towards an Integrated Architecture for Peer-to-Peer and Ad Hoc Overlay Network Applications

L. Yan, K. Sere, X. Zhou, J. Pang

Available in: *Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'04), May 2004. IEEE Computer Society Press.*

Shorten version available as: *L. Yan. MIN: Middleware for Network-Centric Ubiquitous Systems. In IEEE Pervasive Computing, Vol. 3, No. 3, July - September 2004.*

Towards an Integrated Architecture for Peer-to-Peer and Ad Hoc Overlay Network Applications

Lu Yan, Kaisa Sere, Xinrong Zhou
Turku Centre for Computer Science (TUCS) and
Åbo Akademi University
FIN-20520 Turku, FINLAND
{Lu.Yan, Kaisa.Sere, Xinrong.Zhou}@abo.fi

Jun Pang
Centrum voor Wiskunde en Informatica (CWI)
P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands
Jun.Pang@cwi.nl

Abstract

Peer-to-peer (P2P) networks and mobile ad hoc networks (MANET) share some key characteristics: self-organization and decentralization, and both need to solve the same fundamental problem: connectivity. We motivate a study for the convergence of the two overlay network technologies and sketch an evolving architecture towards integrating the two technologies in building overlay network applications.

1. Introduction

Peer-to-peer (P2P) systems are self-organizing, decentralized overlay networks, in which participating nodes contribute resources and cooperate to provide a service. Mobile ad hoc network (MANET) is an autonomous system of mobile hosts (also serving as routers) connected by wireless links, the union of which forms a communication network with arbitrary communication topologies. A P2P network consists of a dynamically changing set of nodes connected via an infrastructure-based network, while a MANET consists of mobile nodes communicating with each other using multi-hop wireless links.

P2P and MANET share some key characteristics: self-organization and decentralization, which lead to a lot of similarities between the two overlay networks [2]:

- **Dynamic topology**. A node in P2P and MANET may join or leave the network at any time and the position of a node in MANET is changing arbitrarily, which leads to no constant routes for any nodes. Both networks have a dynamically changing network topology.
- **Hop connection**. Connections in P2P and MANET are established via exchanging beacon messages only between neighbor nodes. A single hop connection in P2P is typically via TCP links without physical limits, while a single hop in MANET is via wireless links which are usually limited by the radio transmission range.
- **Routing protocol**. Both P2P and MANET routing protocols have to deal with dynamic network topologies due to membership changes or mobility. Typically, a host in P2P and MANET also serves as a router, and employs some flooding-based routing protocols.

The common characteristics shared by P2P and MANET also lead to the same fundamental challenge, that is, how to provide connectivity in a completely decentralized environment. Thus, we are motivated to study the convergence of the two overlay network technologies in terms of the design goals and principles of their routing protocols.

Previously, the P2P and MANET research communities have been working largely in isolation, while facing many common issues like self-organization and decentralization. We argue that it is a promising research direction to bring the two communities together to merge the techniques used in the two areas and perhaps discover unified tricks for the convergence of the two overlay network technologies. As a supporting example, in this paper, we sketch an evolving architecture developed as part of the *Ad Hoc Networking* project at TUCS towards an integrated architecture for peer-to-peer and ad hoc overlay network applications.

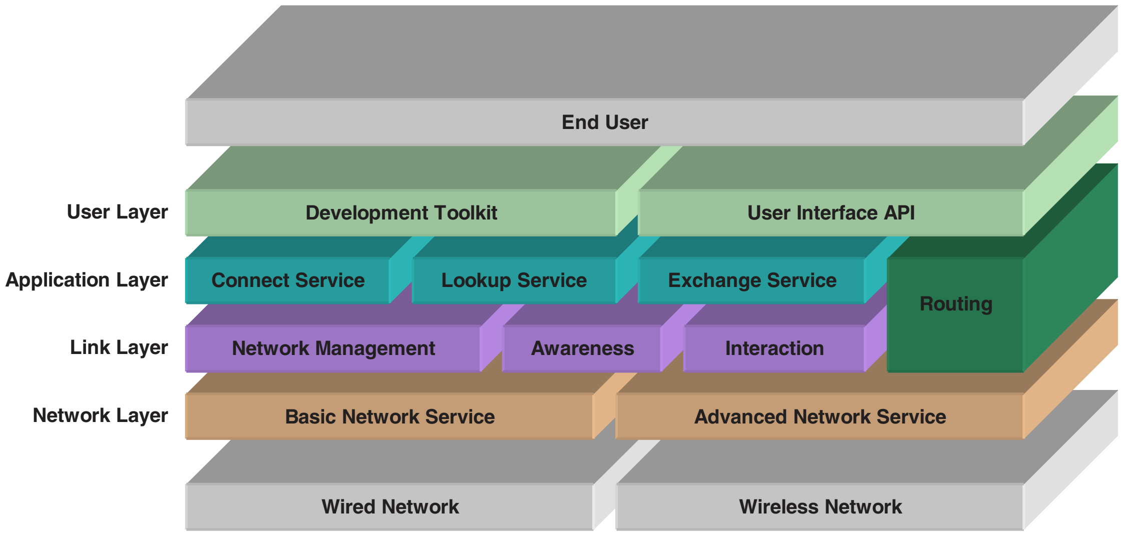


Figure 1: MIN Architecture

The remainder of the paper is organized as follows. We first discuss the research problems of P2P and MANET architectures in the next section. In Section 3, we describe the integrated architecture of P2P and MANET. Section 4 concludes the paper with future research efforts.

2. Research Problems

Today's networks are dependent on wired or wireless infrastructure. This dependence renders the networks vulnerable to disasters and attacks against the fixed infrastructure that supports them. Disasters like floods and earthquakes, as well as wars and terror strikes, can damage or shutdown the whole network. Thus a state-of-the-art research direction of nowadays network is on connectivity.

A network architecture that satisfies the above scenario will be radically different from the current existing network architectures since it cannot rely on a fix infrastructure and dedicated servers. Recent work on P2P overlay networks [5], [6], [7] offers a self-organizing substrate for decentralized network applications. Our general approach is to build a structured P2P overlay with existing technologies upon the basic connectivity provided by MANET in the absence of a dedicated server infrastructure. However, an important challenge is that existing P2P overlay protocols were designed for the Internet, which is a very different environment than MANET. The unique characteristics of this emerging class of networks calls for novel architectures. We present the key challenges as a set of research problems.

- **Self-organizing infrastructure**. Wired networks rely on a fixed infrastructure consisting of routers and DHCP and DNS servers. Any damage or interfere of the server will probably make the whole network out of service. Emerging P2P technologies promise to support self-organizing infrastructure, but these technologies are not directly applicable to the ad hoc wireless environments [3], because they are originally designed for the Internet with constantly stationary nodes, where as nodes are arbitrarily moving in MANET.
- **Decentralized service**. Existing networks depends on dedicated servers providing centralized basic network services like naming, authentication and timing etc. For instance, conventionally there are DHCP and DNS services in a typical network, while supporting this kind of critical network services is beyond the capability of existing P2P networks. Our approach is to build foundations from P2P system, but take advantages of the hierarchical overlay structure contributed by MANET to provide decentralized network services.
- **Integrated routing**. Integrating a P2P routing protocol into a MANET protocol is difficult. P2P overlays in the Internet rely on the IP routing mechanism which is actually application-level routing, while such kind of routing is usually carried out in link-level in MANET [4], [8], [16]. Although typical flooding and multi-hop routing protocols in MANET are peer-to-peer in natural, P2P routing protocols are not directly applicable in MANET due to the namespace problem.

3. The Integrated Architecture

We propose an evolving architecture which is able to provide network connectivity in a decentralized fashion and use self-organizing infrastructures to improve availability of today's network. In this architecture, ad hoc wireless networks can be combined with infrastructure-based networks through ad hoc communications between them. Once basic connectivity is established, hosts could self-organize and cooperatively provide network services that are normally provided by infrastructure servers.

Figure 1 shows the preliminary architecture of a subsystem we are building called the *MIN* that is aimed at addressing a subset of the problems listed in Section 2. The *MIN* is being built on top of an application-level P2P overlay over a link-level MANET, but the architecture is not specific to the implementation environment. We have chosen to focus on two issues, self-organizing infrastructure and integrated routing, which we believe to be fundamental. We feel that decentralized service could be elevated to a higher level of programming abstraction than typical one.

3.1 Application Layer

The *MIN* architecture provides an abstract layer called application layer. This layer is mostly a structured P2P overlay, which is illustrated in Fig. 2.

Previously [1], we have specified a P2P overlay structure in OO-action systems [14]. As an example, you can see the specification of a host.

```
HOST = |[
    attr
      connected := false;
      keyword := NULL;
      target := NULL
    obj
      c: ConnectService;
      l: LookupService;
      e: ExchangeService
    meth
      SetKeyword(k) = Keyword := k;
      SetTarget(t) = target := t
    init
      c := new(ConnectService);
      l := new(LookupService);
      e := new(ExchangeService)
    do
       NOT(connected) →
         connected := c.Connect()
    [] connected AND keyword ≠ NULL →
         target := l.Search(keyword);
         keyword := NULL
    [] connected AND target ≠ NULL →
         e.Exchange(target);
         target := NULL
    od
]|
```

As shown in this specification, three key services are identified in this layer: connect service, lookup service and exchange service [12].

- **Connect service**. A host connects itself to the P2P overlay by establishing a connection with another host currently on the network, and this kind of connection is passed around recursively.
- **Lookup service**. Once a host is connected to the P2P overlay, i.e. it has announced its existence to other members of the P2P overlay, it can then lookup the contents of the P2P overlay. Lookup requests are transmitted in a decentralized manner. One host sends a lookup request to its neighbors, which in turn pass the request along to their neighbors, and so on. Once a host in the P2P overlay has a match, it transmits the hit information back through all the intermediate hosts in the pathway towards the requesting host.
- **Exchange service.** The exchange service can be evoked in an either aggressive or passive manner. Due to the nature of P2P overlay, data are exchanged out-of-network, i.e. a direct exchange between the source and target hosts. Data are never transferred over the P2P overlay.

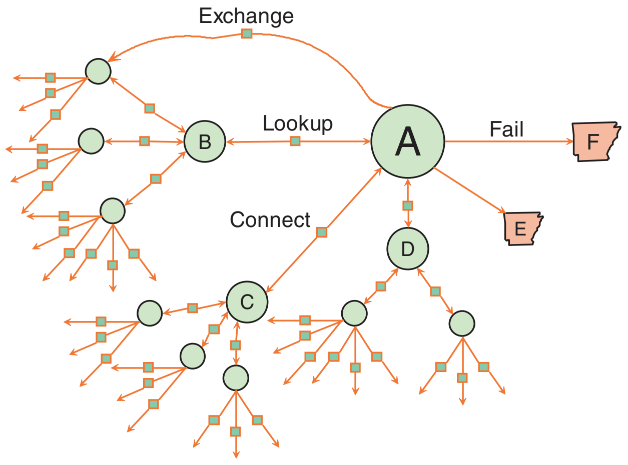


Figure 2: P2P Overlay

3.2 Link Layer

To support ad hoc networking, the *MIN* architecture provides a link layer that allows application-level connections to result in connections to the appropriate logical links, which are either real wired and wireless links or virtual links among hosts. There are three key components in this layer: network management, awareness and interaction.

Previously [9], we have specified this layer in the B method [10]. As an example, you can see the specification of the general context of ad hoc networking environments.

MACHINE *AdHocNet*

SETS
NODES;
CommMSG = { *commMsg*; *routeError* };
RouteMSG = { *routeReq*, *routeRep* }

CONSTANTS
myID

PROPERTIES
$myID \in NODES$

END

In the *AdHocNet* specification, there are three sets: *NODES*, *CommMSG* and *RouteMSG*. They are defined as set of nodes, set of communication messages, and set of routing messages in ad hoc networks. The *CommMSG* has two different elements: *commMsg* and *routeError*. The *commMsg* is used for the communication between nodes. The *routeError* is used when a route is broken. The *RouteMSG* consists of two kinds of routing messages: *routeReq*(route request) and *routeRep*(route reply), which are used in the Awareness component to detect remote nodes. The node's ID *myID* is a constant in the *AdHocNet*, which is an important property of *NODES* to identify a node in networks.

3.2.1 Network Management

Network management is the manager of node connections, which is an important aspect for the MANET design. In general, we consider not only mobility, but also restorability of networking. With this component, a host should be able to set the mode of the node, form, join or leave a network, and manage its connections. As a host is moving arbitrarily, disconnections may happen at any time due to the limited radio transmission range. In order to keep the network working, it is necessary to update the network topology periodically. Moreover, in order to form a self-organizing network, and support multi-hop routing in forwarding packages, it is necessary to have the network manager in every host.

In the specification of network management, there are three components: *netManager*, *modeSet* and *Connector*. The relationship of components is shown in Fig. 3.

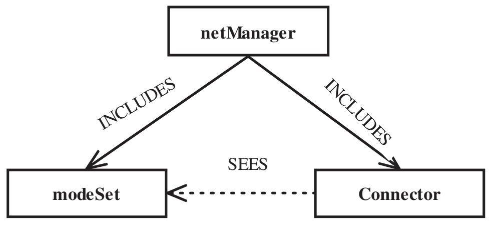


Figure 3: Network Management

The *netManager* includes *modeSet* and *Connector*, and uses their operations. In ad hoc networking, it manages activities of the system and updates connections to neighbor nodes. There are two modes that can be set in *modeSet*, either discoverable or non-discoverable. When a node is in discoverable mode, it can be discovered by other nodes in the network. Otherwise, it cannot be discovered if it is in non-discoverable mode. The *Connector* is used when a node wants to connect or disconnect its neighbors. For instance, in the network setup stage, a node can join into networks by connecting its neighbor nodes which are in discoverable mode. Once the connections are ready, the network is established. When shutting down the system, the netManager disconnects all the connections to neighbor nodes and sets the node into non-discoverable mode.

3.2.2 Awareness

Awareness of mobile computing is used to sense a certain environment in order to present and update context of mobile systems [11]. In our system, we focus on detecting local and remote nodes and processing incoming messages. The awareness of our system is divided into two parts.

- **Node awareness.** There are two kinds of awareness of node detections: local awareness – system can detect local nodes within the radio transmission range; remote awareness – system should also be able to detect a friendly remote node whose ID is already known. In local awareness, a node detects its neighbor nodes and the detected nodes will be connected and used to update topologies in Network Management. In remote awareness, a node tries to find out friendly remote nodes with known IDs and possible routes for communication.

- **Message awareness**. In message awareness, a node processes incoming messages according to the format of data packets. There are two kinds of messages in networking: communication message and routing message. In our specification, the communication message is used for communication between nodes, and there are two types of routing message: route request message and route reply message. Due to the different types of messages, the processing will be different. As shown in Fig. 4, if the received message is a communication message, the system will check the packet head, and receive or forward this message depending on the next hop ID on the route. In case this ID is unrecognizable, the system will report a broken route. If the incoming message is a routing message, the system will process this message according to the routing protocols in our system.

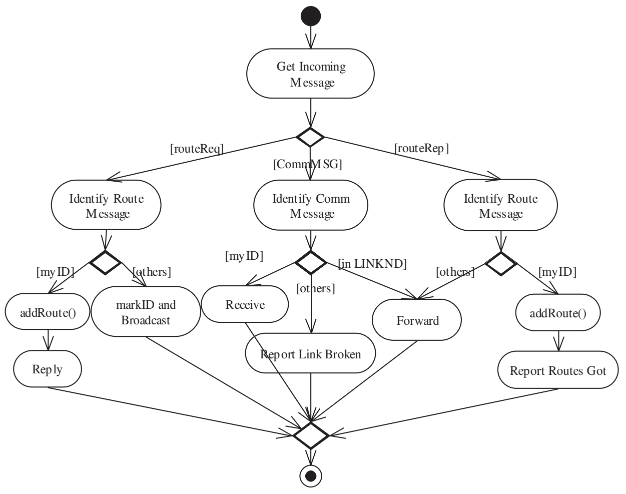


Figure 4: Incoming Message Processing

3.2.3 Interaction

Interaction mainly concerns communication links between nodes. We consider an opening session for interactive communication between nodes. In such a session, the source and destination nodes can send and receive messages and update routing information for communication.

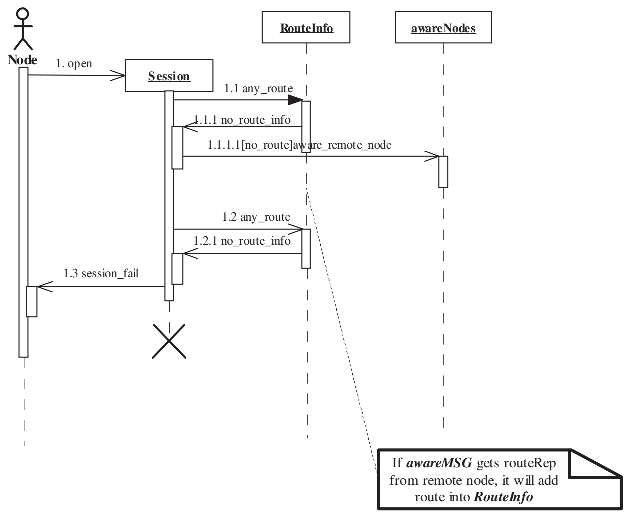


Figure 5: Opening Session

As shown in Fig. 5, when the system opens such a session and starts interactive communication, the source node will select a route from the routing table or detect a new route to reach the destination node. If there is no available route or the destination node is not detected in the network, the opening session fails and a failure message is sent back to the source node. In successful case, once a route is available, a communication session between the source node and destination node is created and the interactive communication starts.

In the interactive communication, topologies might be changed and it will lead to route breaks or changes. Thus the route maintenance and recovery are needed for interactive communication. Figure 6 shows how the route is recovered when the system knows that the route is broken. In our design, it is assumed that multiple routes discovery protocols are used. For example, when source node S is communicating with destination node D, S sends data packets to D along with the selected route. During their communication, if S gets to know that the communication route is broken, S doesn't need to rediscover a new route immediately because S might have detected several routes in the previous discovery. It can then choose another available route and replace the broken one. If none of the routes reaches to the destination, the system will start route discovery again.

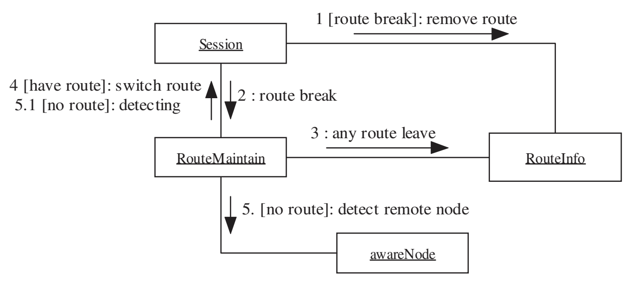

1 [route break]: remove route
Session
4 [have route]: switch route
5.1 [no route]: detecting
2 : route break
3 : any route leave
RouteMaintain
RouteInfo
5. [no route]: detect remote node
awareNode

Figure 6: Route Recovery

3.3 Integrated Routing

The primary challenge with using a P2P routing protocol in MANET is the fact that P2P overlays in the wired Internet rely on the IP routing infrastructure to perform hop-by-hop routing between neighbor nodes in the overlay. Thus the key problem in the integration is that P2P overlay routing protocols run in a logical namespace but MANET routing protocols run in a physical namespace. A possible solution to the integration is to build a one-to-one mapping between the IP address of the mobile nodes and their node IDs in the namespace, and replace the routing table entries which used to store IP addresses with source routes.

For instance, to integrate a Gnutella-like [12] P2P protocol into a DSR-like [15] MANET protocol, unique node IDs are first assigned to nodes in a MANET as is done in P2P overlay on top of the Internet. Node IDs can be generated by hashing the IP addresses of the hosts using collision-resistant hashing functions like SHA-1 [13], thus obtaining a unique node ID for each node in the network. The mobile nodes in the ad hoc network can then form a P2P overlay in the same fashion as in the Internet. Nodes can handle join, leave and fail actions in a similar way as before. The structure of the routing states is also similar as before, with one exception: the routing table stores the source route to reach the destination node ID, not just a simple IP address. To route a data packet, a message key is first generated by hashing the destination IP address, and then the message is routed in the overlay similarly to in the overlay on top of the Internet. The only difference is that each overlay hop in ad hoc networks is a multi-hop source route, while each overlay hop in the Internet is a multi-hop IP route.

4. Conclusions and Future Work

The main contribution of this work is that it proposes a novel architecture, integrating P2P and MANET technologies together, to reduce the dependence of networking on wired and wireless infrastructure, thus extending the reachability of nowadays networks and increasing their resilience to disasters and attacks. Another contribution of this work is that it is the first architecture-centric approach for the construction of overlay network applications that allows us to define a unified networking environment, taking advantages from both P2P and MANET technologies.

The work presented in this paper is in its early stages. At present we are evaluating the initial version of the *MIN* framework through analysis, simulation, and a prototype implementation. In the formal respect, a complete formal specification of the architecture is being underway. As a future work, in particular, performance modeling and evaluation of integrating P2P and MANET routing protocols will be undertaken. A further future work of our research is to implement a middleware in this integrated architecture for use by developers of overlay network applications.

Acknowledgements

The authors are grateful for Jincheng Ni and Nayyar Iqbal's contribution to this project.

References

[1] L. Yan and K. Sere, *Stepwise Development of Peer-to-Peer Systems*, Proceedings of the 6th International Workshop in Formal Methods (IWFM'03), Dublin, Ireland, July 2003. Electronic Workshops in Computing (eWiC), British Computer Society (BCS).

[2] Y. C. Hu, S. M. Das, and H. Pucha, *Exploiting the synergy between peer-to-peer and mobile ad hoc networks.* In Proceedings of HotOS-IX: 9th Workshop on Hot Topics in Operating Systems, May 2003.

[3] C. Blake and R. Rodrigues, *High availability, scalable storage, dynamic peer network: Pick two*. In Proc. HotOS IX, Kauai, Hawaii, May 2003.

[4] M. Castro, P. Druschel, A. -M. Kermarrec, and A. Rowstron, *SCRIBE: A large-scale and decentralized application-level multicast infrastructure*. IEEE JSAC, 20(8), Oct. 2002.

[5] A. Muthitacharoen, R. Morris, T. Gil, and B. Chen, *Ivy: A read/write peer-to-peer file system*. In Proc. of the 5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, MA, December 2002.

[6] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, *Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination.* In NOSSDAV, June 2001.

[7] P. F. Tsuchiya, *Landmark hierarchy: a new hierarchy for routing in very large networks*. In Proc. of MobiHOC, Standford, CA, Aug. 2000.

[8] S. Ratnasamy, M. Handley, R. Karp, and S.Shenker, *Application-level multicast using content-addressable networks*, In NGC, Nov. 2001.

[9] L. Yan, J. Ni and K. Sere, *Towards a Systematic Design for Ad hoc Network Applications*, Proceedings of the 15th Nordic Workshop on Programming Theory (NWPT'03), Oct. 2003.

[10] E. Sekerinski and K. Sere (Eds), *Program Development by Refinement: Case Studies Using the B Method*, Springer-Verlag, 1999.

[11] T. Selker and W. Burleson, *Context-aware Design and Interaction in Computer System*, IBM Systems Journal, Vol. 39, No. 3 & 4, 2000.

[12] I. Ivkovic, *Improving Gnutella Protocol: Protocol Analysis and Research Proposals*, Technical reports, LimeWire LLC, 2001.

[13] FIPS 180-1, *Secure Hash Standard*, Technical Report Publication 180-1, Federal Information Processing Standard (FIPS), NIST, US Department of Commerce, Washington D.C., April 1995.

[14] M. Bonsangue, J. N. Kok and K. Sere, *An approach to object-orientation in action systems*, Proceedings of Mathematics of Program Construction (MPC'98), Martstrand, Sweden, June 1998. LNCS 1422, Springer Verlag.

[15] D. B. Johnson and D. A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks.* In T. Imielinski and H. Korth (Eds), Mobile Computing, chap. 5, p. 153-181, Kluwer Academic Publishers, 1996.

[16] W. Adjie-Winoto, E. Schwartz and H. Balakrishnan, *An Architecture for Intentional Name Resolution and Application-level Routing*. MIT Technical Reports, TR-775, 1999.

Paper VI

Performance Evaluation and Modeling of Peer-to-Peer Systems over Mobile Ad hoc Networks

L. Yan

Submitted to *Performance Evaluation.*

Previous version available as: *L. Yan. Performance Evaluation and Modeling of Peer-to-Peer Systems over Mobile Ad hoc Networks. TUCS Technical Reports, No. 678, Turku Centre for Computer Science, Finland, 2005.*

Shorten version available as: *L. Yan. Performance Modeling of Mobile P2P Systems. In Proceedings of the International Conference on Computer Networks and Mobile Computing (ICCNMC'05), August 2005. Lecture Notes in Computer Science 3619, Springer-Verlag.*

Performance Evaluation and Modeling of Peer-to-Peer Systems over Mobile Ad hoc Networks

Lu Yan
Åbo Akademi University, Department of Computer Science

TUCS Technical Report
No 678, March 2005

Abstract

With the advance in mobile wireless communication technology and the increasing number of mobile users, peer-to-peer computing, in both academic research and industrial development, has recently begun to extend its scope to address problems relevant to mobile devices and wireless networks. This paper is a performance study of peer-to-peer systems over mobile ad hoc networks. We show that cross-layer approach performs better than separating the overlay from the access networks with the comparison of different settings for the peer-to-peer overlay and underlaying mobile ad hoc network. We then present a performance model which captures most facets of mobile peer-to-peer systems. We hope our results would potentially provide useful guidelines for mobile operators, value-added service providers and application developers to design and dimension mobile peer-to-peer systems.

Keywords: Mobile, Peer-to-Peer, Performance, Modeling, Evaluation

TUCS Laboratory
Distributed Systems Design Laboratory

1. Introduction

Peer-to-Peer (P2P) computing is a networking and distributed computing paradigm which allows the sharing of computing resources and services by direct, symmetric interaction between computers. With the advance in mobile wireless communication technology and the increasing number of mobile users, peer-to-peer computing, in both academic research and industrial development, has recently begun to extend its scope to address problems relevant to mobile devices and wireless networks.

Mobile Ad hoc Networks (MANET) and P2P systems share a lot of key characteristics: self-organization and decentralization, and both need to solve the same fundamental problem: connectivity. Although it seems natural and attractive to deploy P2P systems over MANET due to this common nature, the special characteristics of mobile environments and the diversity in wireless networks bring new challenges for research in P2P computing.

Currently, most P2P systems work on wired Internet, which depends on application layer connections among peers, forming an application layer overlay network. In MANET, overlay is also formed dynamically via connections among peers, but without requiring any wired infrastructure. So, the major differences between P2P and MANET in this paper are (a) P2P is generally referred to the *application layer*, but MANET is generally referred to the *network layer*, which is a lower layer concerning network access issues. Thus, the immediate result of this layer partition reflects the difference of the packet transmission methods between P2P and MANET: the P2P overlay is a unicast network with *virtual* broadcast consisting of numerous single unicast packets; while the MANET overlay always performs *physical* broadcasting. (b) Peers in P2P overlay are usually referred to *static* nodes though no priori knowledge of arriving and departing is assumed, but peers in MANET are usually referred to *mobile* nodes since connections are usually constrained by physical factors like limited battery energy, bandwidth, computing power, etc.

The above similarities and differences between P2P and MANET lead to an interesting but challenging research on P2P systems over MANET. In fact, this scenario seems feasible and promising, and possible applications include car-to-car communication in a field-range MANET, an e-campus system for mobile e-learning applications in a campus-range MANET on top of IEEE 802.11, and a small applet running on mobile phones or PDAs enabling mobile subscribers exchange music, ring tones and video clips via Bluetooth, etc.

This paper is a performance study of peer-to-peer systems over mobile ad hoc networks. In the following section we will review previous work on P2P and MANET. After comparing different settings for the peer-to-peer overlay and underlaying mobile ad hoc network, we show that cross-layer approach performs better than separating the overlay from the access networks in section 3. In section 4, we present a performance model which captures most facets of mobile peer-to-peer systems. In section 5, we apply our analytical model to practical network design problems and analyze some important QoS issues. Finally, section 6 concludes the paper.

2. Background and State-of-the-Art

Since both P2P and MANET are becoming popular only in recent years, the research on P2P systems over MANET is still in its early stage. The first documented system is Proem [1], which is a P2P platform for developing mobile P2P applications, but it seems to be a rough one and only IEEE 802.11b in ad hoc mode is supported. 7DS [2] is another primitive attempt to enable P2P resource sharing and information dissemination in mobile environments, but it is rather a P2P architecture proposal than a practical application. In a recent paper [3], Passive Distributed Indexing was proposed for such kind of systems to improve the search efficiency of P2P systems over MANET, and in ORION [4], a Broadcast over Broadcast routing protocol was proposed. The above works focus on either P2P architecture or routing schema design, but how efficient is the approach and what is the performance experienced by users are still in need of further investigation.

Previous work on performance study of P2P over MANET is mostly based on the simulative approach and no concrete analytical model is introduced. Performance issues of this kind of systems are first discussed [5] with experiment results. There is a survey of such kind of systems [6] but no further conclusions were derived. There are also some sophisticated experiments and discussions [7] on P2P communication in MANET. Recently, B. Bakos etc. with Nokia Research analyzed a Gnutella-style protocol query engine on mobile networks with different topologies [8], and T. Hossfeld etc. with Siemens Labs conducted a simulative performance evaluation of mobile P2P file-sharing [9]. However, all above works fall into practical experience report category and no performance models are proposed.

We believe that to understand the performance issues, rigorous analytical models are needed, which capture the relation between key system parameters and performance metrics. In the remaining sections we present our efforts on performance evaluation of mobile peer-to-peer systems, especially from users' point of view, e.g. what is the performance experience of a user in mobile P2P systems? We then present a performance model which captures most facets of mobile peer-to-peer systems. We hope our results would potentially provide useful guidelines to design and dimension mobile peer-to-peer systems.

3. Performance Evaluation of P2P over MANET

As stated before, we, in this paper, focus only on the performance of P2P systems over MANET from users' point of view since it makes greater impact on the design decisions of such kind of system for mobile operators, value-added service providers and application developers. Specifically, we want to answer the following questions: (1) How can we perform an efficient search in mobile P2P systems? (2) and what is the performance experience when many users try to retrieve data with parallel downloading scheme? (We leave the answer to the second question to section 4 and 5.) To answer the first question, the routing protocols and route discovery efficiency of different settings for the peer-to-peer overlay and underlaying mobile ad hoc network should be further investigated.

There are many routing protocols in P2P networks and MANET respectively. For instance, one can find a very substantial P2P routing scheme survey from HP Labs [10], and US Navy Research publish ongoing MANET routing schemes [11]; but all above schemes fall into two basic categories: broadcast-like and DHT-like. More specifically, most early P2P search algorithms, such as in Gnutella [12], Freenet [13] and Kazaa [14], are broadcast-like and some recent P2P searching, like in eMule [15] and BitTorrent [16], employs more or less some feathers of DHT. On the MANET side, most on-demand routing protocols, such as DSR [17] and AODV [18], are basically broadcast-like. Therefore, we here introduce different approaches to integrate these protocols in different ways according to categories.

3.1. Broadcast over Broadcast

A rudimental approach is to employ a broadcast-like P2P routing protocol at the application layer over a broadcast-like MANET routing protocol at the network layer. Intuitively, in these settings, every routing message broadcasting to the *virtual* neighbors at the application layer will result to a full broadcasting to the corresponding *physical* neighbors at the network layer.

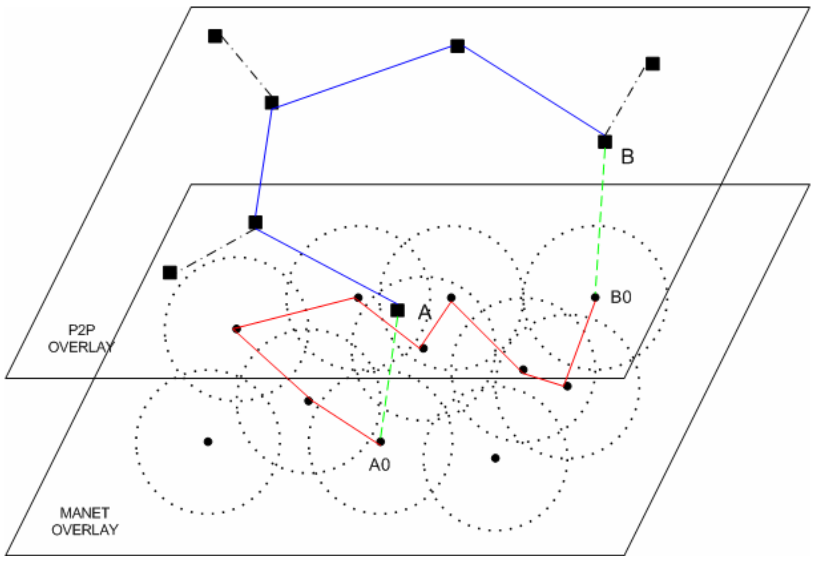


Figure 1. Broadcast over Broadcast

The scheme is illustrated in Figure 1 with a searching example: peer *A* in the P2P overlay is trying to search for a particular piece of information, which is actually available in peer *B*. Due to the broadcast mechanism, the search request is transmitted to *A*'s neighbors, and recursively to all the members in the network, until a match is found or timeout. There is a blue line representing the routing path at the application layer. Then we map this searching process into the MANET overlay, where node *A0* is the corresponding mobile node to the peer *A* in the P2P overlay, and *B0* is related to *B* in the same way. Since the MANET overlay also employs a broadcast-like routing protocol, the request from node *A0* is flooded (broadcast) to its directly connected neighbors, which themselves flood their neighbors etc., until the request is answered or a maximum number of flooding steps occur. The route establishing lines in that network layer is highlighted in red, where we can find that there are few overlapping routes between these two layers though each of them employs a broadcast-like protocol.

We have studied Guntella [19], a typical broadcast-like P2P protocol [20]. This is a pure P2P protocol, as shown in Figure 2, in which no advertisement of shared resources (e.g. directory or index server) occurs. Instead, each request from a peer is broadcasted to its directly connected peers, which themselves broadcast this request to their directly connected peers etc., until the request is answered or a maximum number of broadcast steps occur. It is easy to see that this protocol requires a lot of network bandwidth, and it does not prove to be very scalable. The complexity of this routing algorithm is *O*(*n*) [21, 22].

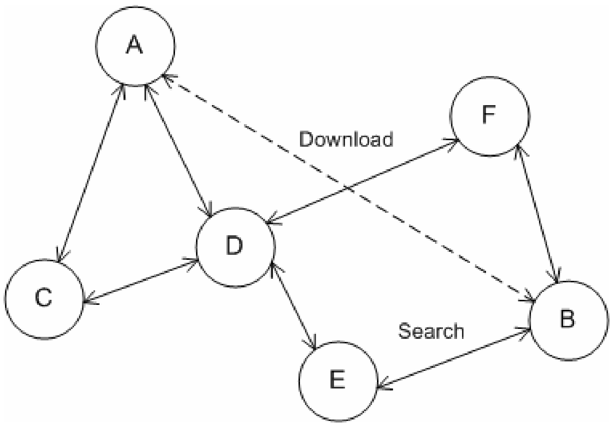


Figure 2. Broadcast-like P2P Protocol

Generally, most on-demand MANET protocols, like DSR [23] and AODV [24], are broadcast-like in nature [25]. Previously, we have studied AODV, one typical broadcast-like MANET protocol [26]. As shown in Figure 3, in that protocol, each node maintains a routing table only for active destinations: when a node needs a route to a destinations, a path discovery procedure is started, based on a RREQ (route request) packet; the packet will not collect a complete path (with all IDs of involved nodes) but only a hop count; when the packet reaches a node that has the destination in its routing table, or the destination itself, a RREP (route reply) packet is sent back to the source (through the path that has been set-up by the RREQ packet), which will insert the destination in its routing table and will associate the neighbour from which the RREP was received as preferred neighbour to that destination. Simply speaking, when a source node wants to send a packet to a destination, if it does not know a valid route, it initiates a route discovery process by flooding RREQ packet through the network. AODV is a pure on-demand protocol, as only nodes along a path maintain routing information and exchange routing tables. The complexity of this routing algorithm is *O*(*n*) [27].

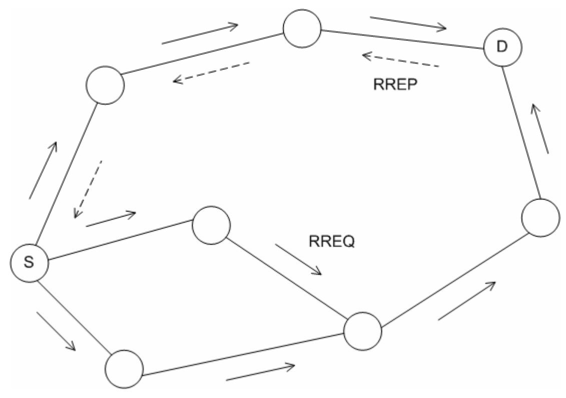


Figure 3. Broadcast-like MANET Protocol

This approach is probably the easiest one to implement, but the drawback is also obvious: the routing path of the requesting message is not the shortest path between the source and destination (e.g. the red line in Figure 1), because *virtual* neighbors in the P2P overlay are not necessarily *physical* neighbors in the MANET overlay, and actually these nodes might be physically far away from each other. Therefore, the resulting routing algorithm complexity of this broadcast over broadcast scheme is unfortunately $O(n^2)$ though each layer's routing algorithm complexity is $O(n)$ respectively.

It is not practical to deploy such kind of scheme for its serious scalability problem due to the double broadcast. Taking the energy consumption portion into consideration, which is critical to mobile devices, the double broadcast will also cost a lot of energy, and make it infeasible in cellular wireless networks.

3.2. DHT over Broadcast

The scalability problem of broadcast-like protocols has long been observed and many revisions and improvement schemas are proposed [28, 29, 30]. To overcome the scaling problems in broadcast-like protocols where data placement and overlay network construction are essentially random, there are a number of proposals on structured overlay designs. Distributed Hash Table (DHT) [31] and its varieties [32, 33, 34] advocated by Microsoft Research seem to be promising routing algorithms for overlay networks. Therefore it is interesting to see the second approach: to employ a DHT-like P2P routing protocol at the application layer over a broadcast-like MANET routing protocol at the network layer.

The scheme is illustrated in Figure 4 with the same searching example. Compared to the previous approach, the difference lies in the P2P overlay: in a DHT-like protocol, files are associated to keys (e.g. produced by hashing the file name); each node in the system handles a portion of the hash space and is responsible for storing a certain range of keys. After a lookup for a certain key, the system returns the identity (e.g. the IP address) of the node storing the object with that key. The DHT functionality allows nodes to put and get files based on their key, and each node handles a portion of the hash space and is responsible for a certain key range. Therefore, routing is location-deterministic distributed lookup (e.g. the blue line in Figure 4).

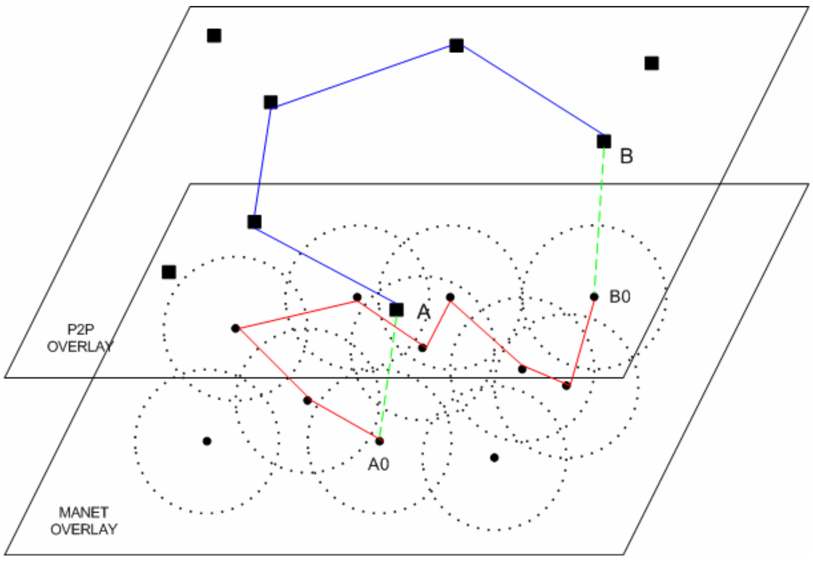


Figure 4. DHT over Broadcast

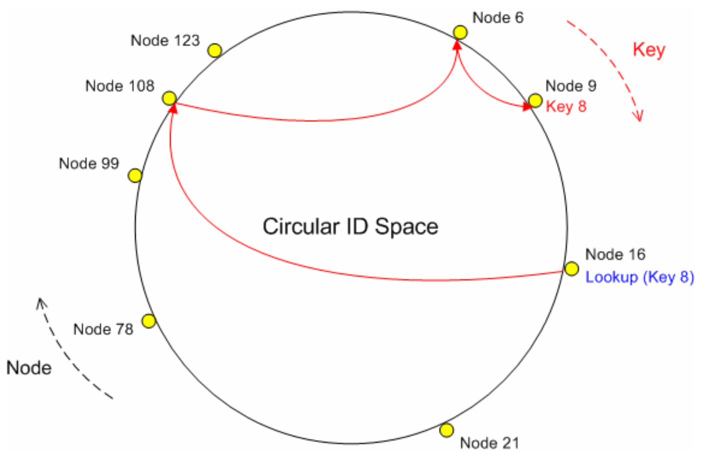


Figure 5. DHT-like P2P Protocol

DHT was first proposed by Plaxton [35], and soon proved to be a useful substrate for large distributed systems. A number of projects are proposed to build Internet-scale facilities layered above DHTs, among them are Chord [31], CAN [32], Pastry [33], Tapestry [34] etc. As illustrated in Figure 5, all of them take a key as input and route a message to the node responsible for that key. Nodes have identifiers, taken from the same space as the keys. Each node maintains a routing table consisting of a small subset of nodes in the system. When a node receives a query for a key for which it is not responsible, the node routes the query to the *hashed* neighbor node towards resolving the query. In such a design, for a system with n nodes, each node has $O(log\ n)$ neighbors, and the complexity of the DHT-like routing algorithm is $O(\log n)$ [36].

Additional work is required to implement this approach, partly because DHT requires periodical maintenance (i.e. it is just like an Internet-scale hash table, or a large distributed database): since each node maintains a routing table (i.e. hashed keys) to its neighbors according to the DHT algorithm, following a node join or leave, there is always a nearest key reassignment between nodes.

DHT over Broadcast approach is obviously better than the previous one, but it still does not solve the shortest path problem as in the Broadcast over Broadcast scheme. Though the P2P overlay algorithm complexity is optimized to $O(log\ n)$, the mapped message routing in the MANET overlay is still in the broadcast fashion with complexity $O(n)$; the resulting algorithm complexity of this approach is as high as $O(n\ log\ n)$.

This approach still requires a lot of network bandwidth, and hence does not prove to be very scalable, but could be efficient in limited communities, such as a company network.

3.3. Cross-Layer Routing

A further step of the Broadcast over Broadcast approach would be a Cross-Layer Broadcast. Due to the similarity of Broadcast-like P2P and MANET protocols, the second broadcast could be skipped if the peers in the P2P overlay would be mapped directly into the MANET overlay, and the result of this approach would be the merge of application layer and network layer (i.e. the *virtual* neighbors in P2P overlay overlaps the *physical* neighbors in MANET overlay).

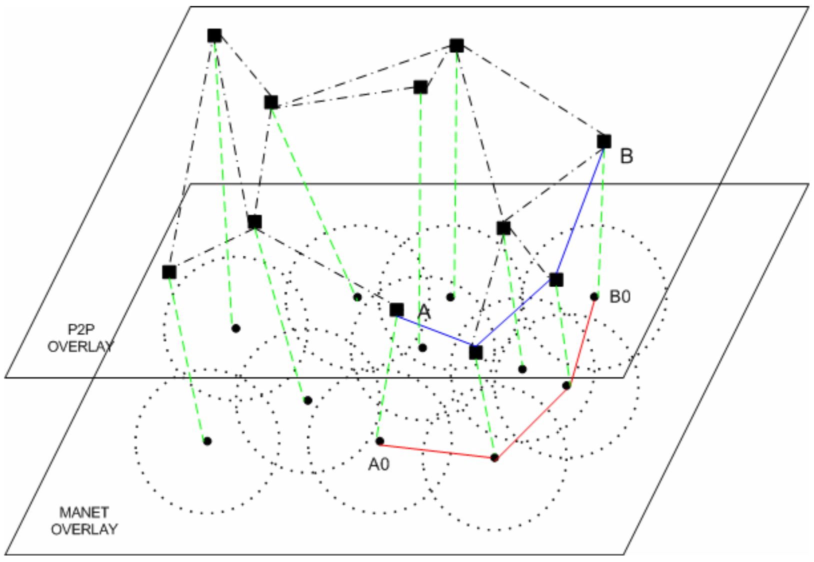


Figure 6. Cross-Layer Broadcast

The scheme is illustrated in Figure 6, where the advantage of this cross-layer approach is obvious: the routing path of the requesting message is the shortest path between source and destination (e.g. the blue and red lines in Figure 6), because the *virtual* neighbors in the P2P overlay are *de facto physical* neighbors in the MANET overlay due to the merge of two layers. Thanks to the nature of broadcast, the algorithm complexity of this approach is $O(n)$, making it suitable for deployment in relatively large scale networks, but still not feasible for Internet scale networks.

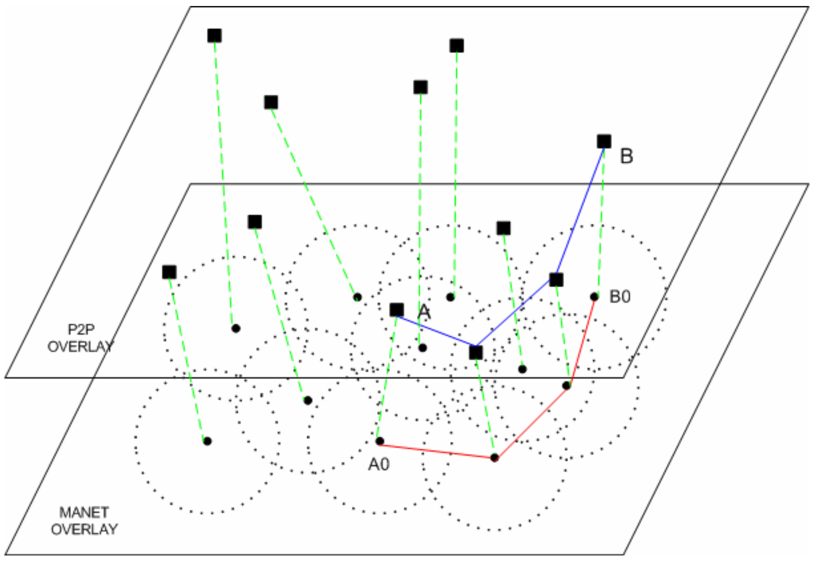


Figure 7. Cross-Layer DHT

It is also possible to design a Cross-Layer DHT in Figure 7 with the similar inspiration, and the algorithm complexity would be optimized to $O(log\ n)$ with the merit of DHT, which is advocated to be efficient even in Internet scale networks. The difficulty in that approach is implementation: there is no off-the-shelf DHT-like MANET protocol as far as we know, though recently, some research projects, like Ekta [37], towards a DHT substrate in MANET are proposed.

As an answer to Question 1, we show the cross-layer approach performs better than separating the overlay from the access networks, with the comparison of different settings for the peer-to-peer overlay and underlaying mobile ad hoc network in above four approaches in Table 1.

Table 1. How efficient does a user try to find a specific piece of data?

	Efficiency	Scalability	Implementation
Broadcast over Broadcast	$O(n^2)$	N.A.	Easy
DHT over Broadcast	$O(n\ log\ n)$	Bad	Medium
Cross-Layer Broadcast	$O(n)$	Medium	Difficult
Cross-Layer DHT	$O(log\ n)$	Good	N.A.

4. Modeling Download Performance

The download performance modeling is a relatively new issue compared to the search performance modeling, which was already extensively studied in some P2P and MANET research [38, 39, 40]. In this section, we present our efforts towards a performance model of downloading in such kind of systems, and thus answer Question

2: “what is the performance experience when many users try to retrieve data with parallel downloading scheme?”

4.1. Preliminary Assumptions

Though early research on modeling has mainly focused on routing performance and searching efficiency, recently, there were some works on modeling the download performance. The Markov chain approach has been brought forth [41] for a queue system model and some measurement studies were mentioned [42]; more recently, Stochastic fluid models are studied [43, 44, 45], which provide a more intuitive and deterministic approach. Our work uses the same approach as [45, 46]; but taking the idea into mobile environments, more realistic scenarios and physical constraints should be introduced, and old notions should have new interpretations.

Since the introduction of Tornado Code [47, 48] has been a popular technique on recently parallel downloading systems, here we assume: (1) the parallel download process in our model is Tornado-like, which reduces the requirement for coordination and signalling. Due to the limited bandwidth of existing wireless networks (probably accompanied with expensive data transmission charge, e.g. cellular network), (2) it is reasonable to allow the *pure downloader* (i.e. *leech*) exist in the system. Therefore, as illustrated in Figure 8, there are three types of peers in our model: (a) normal peer (i.e. *contributor*), which owns part of the file (i.e. ordinary downloader), but still allows others to download from itself. This type is the most common one and it actually constitutes the majority in our system. (b) pure downloader (i.e. *leech*), which just downloads but never uploads. The realistic implication of this type may be physically constrained mobile devices (e.g. cellular phones with limited bandwidth or associated with too expensive data transmission charge). (c) pure uploader (i.e. *seed*), which already have all pieces of the file but still stays in the system to allow others to download from itself. The realistic implication of that type may be content publishers (e.g. mobile operator’s service point).

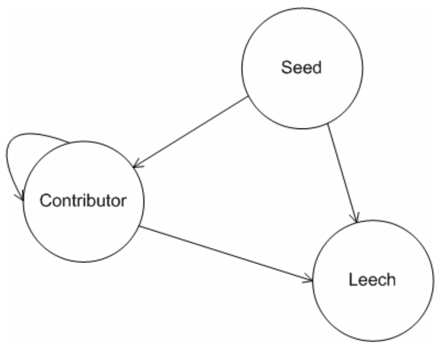


Figure 8. Three Types of Peers

Although there is heterogeneity in realistic infrastructure [49], such as bandwidth, latency, availability, etc., here we make a trade-off between the simplicity of the model and its ability to capture all facets, and assume (3) all peers in our model have equal capacity (i.e. all peers have the same upload and download bandwidth). With the above assumptions and the parameters in Table 2, we can derive that at time *t*, there are $\beta\, x(t)$ leeches and $(1-\beta)\, x(t)$ contributors in our system.

Table 2. Parameters Used in the Model

Parameter	Meaning
$x(t)$	Number of downloaders (i.e. contributors and leeches) at time *t*
β	Selfish rate (i.e. leech portion)
$y(t)$	Number of seeds at time *t*
λ	Arrival rate of new download request (Possion process)
μ	Upload bandwidth of each peer
τ	Download bandwidth of each peer
ρ	Abort rate of downloaders
κ	Leave rate of seeds

4.2. The Model

The queue-like model of one peer in our system is illustrated in Figure 9. As noted here, during the download and upload process, it is also possible that peers will get offline or abort the process, and in order to make the model simple, here we use abort rate ρ and leave rate κ to model these interrupted processes.

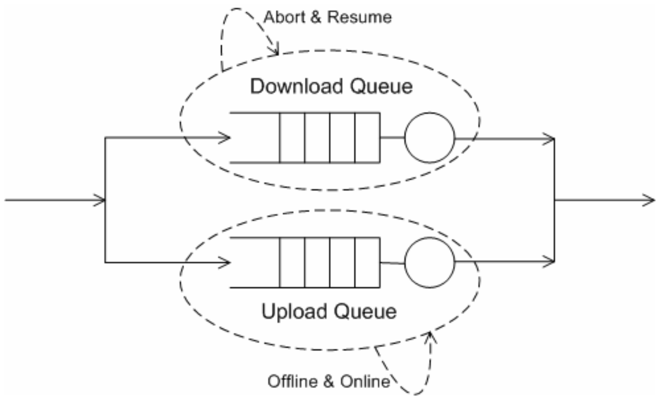


Figure 9. Queue-Like Model of One Peer

In a P2P download and upload scheme, it is natural to expect more on the download side (i.e. this implies $\tau \geq \mu$); so taken the download bandwidth constraint into account, the total upload bandwidth should be $min(\mu((1-\beta)\, x(t) + y(t)), \tau\, x(t))$, and the arrival and departure rate of download request will be λ and $min(\mu((1-\beta)\, x(t) + y(t)), \tau\, x(t)) + \rho\, x(t))$ respectively. The arrival and departure rate of upload request will be $min(\mu((1-\beta)\, x(t) + y(t)), \tau\, x(t))$ and $\kappa\, y(t)$. Thus the fluid model is derived as

$$\frac{d}{dt}x(t) = \lambda - \min[\mu[(1-\beta)x(t) + y(t), \tau x(t)]] - \rho x(t)$$

$$\frac{d}{dt}y(t) = \min\left[\mu\left[(1-\beta)x(t) + y(t), \tau x(t)\right]\right] - \kappa y(t)$$

In a steady state, the number of downloaders and seeds should be independent of time (i.e. $d(x(t))/dt = d(y(t))/dt = 0$); and then if we define

$$\frac{1}{\iota} = \frac{1}{1-\beta}\cdot\left(\frac{1}{\mu} - \frac{1}{\kappa}\right)$$

where ι can be interpreted as *effective* upload bandwidth compared to *nominal* upload bandwidth μ (i.e. after considering the impact of leeches), those equations can be solved as

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \dfrac{\lambda}{\tau\left(1+\dfrac{\rho}{\tau}\right)} \\ \dfrac{\lambda}{\kappa\left(1+\dfrac{\rho}{\tau}\right)} \end{pmatrix} \quad \text{when} \quad \frac{1}{\tau} \geq \frac{1}{\iota}$$

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \dfrac{\lambda}{\iota\left(1+\dfrac{\rho}{\iota}\right)} \\ \dfrac{\lambda}{\kappa\left(1+\dfrac{\rho}{\iota}\right)} \end{pmatrix} \quad \text{when} \quad \frac{1}{\tau} < \frac{1}{\iota}$$

where the limited download bandwidth and limited upload bandwidth is the constraint respectively. Furthermore, if we define

$$\frac{1}{\phi} = \max\left(\frac{1}{\tau}, \frac{1}{\iota}\right)$$

where φ can be interpreted as *bottleneck* bandwidth intuitively, we obtain the solution as

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} \dfrac{\lambda}{\phi\left(1+\dfrac{\rho}{\phi}\right)} \\ \dfrac{\lambda}{\kappa\left(1+\dfrac{\rho}{\phi}\right)} \end{pmatrix}$$

Finally, we derive the average download time for a peer with Little's Law [50]

$$\text{Time} = \frac{1}{\phi + \rho} \quad \text{where} \quad \frac{1}{\phi} = \max\left(\frac{1}{\tau}, \frac{1}{\iota}\right)$$

5. Performance Analysis with the Model

In the model presented in the previous section, it is clear that different settings of β, μ, τ, ρ and κ will lead to different performance; so in this section we will use our analysis model to provide some insights in the network.

5.1. Selfish Peers

For a fixed set of network parameters, we first study the impact of β on the network performance. The realistic interpretation of β is interesting, which is somehow related to peer strategy and incentive mechanism (i.e. *selfish* peers or *leeches*).

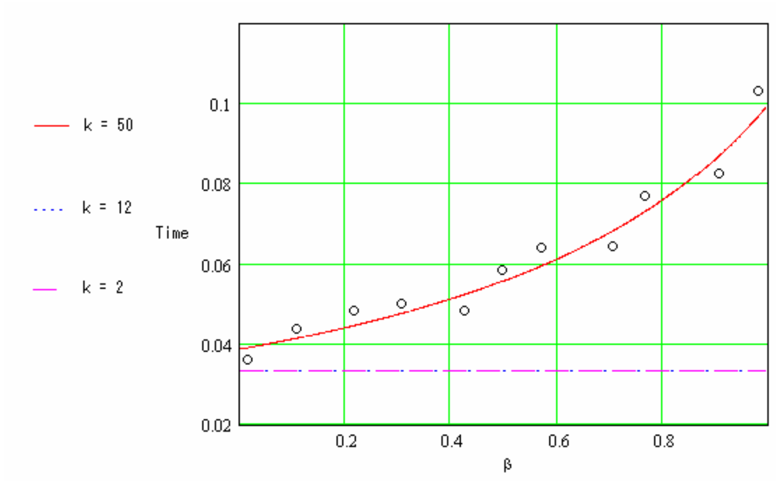


Figure 10. Impact of β on Network Performance

The network parameters we have chosen are: μ = 12kbps, τ = 20kbps, ρ = 10kbps, κ_0 = 50kbps, κ_1 = 12kbps, κ_2 = 2kbps. In this scenario, we consider the effect of *selfish* peers. Intuitively, the existing *leeches* will degrade the system performance because they just download from others and never upload. The red curve in Figure 10 for κ_0 = 50kbps justifies our intuition.

From the observation, it is obvious that *Time* is a non-decreasing function of β. We can also find the upper bound and lower bound of *Time* if we consider two extreme cases: $\beta = 1$ (i.e. all downloaders are *selfish* and no one uploads to others) and $\beta = 0$ (i.e. there is no *leeches* in the system).

At this point, we are all happy with our intuition; but if we change the value of κ into κ_1 = 12kbps and κ_2 = 2kbps, something strange happens. As shown in Figure 10 as two overlapped horizontal lines, the network performance is constant, independent of β. We briefly comment on this situation: recall the *bottleneck* bandwidth definition in the previous section, it actually means the downloading bandwidth is the bottleneck since $\mu \geq \kappa$; in such a situation, the *leeches* make no harm to the system since the whole system performance is constrained by the limited download speed (i.e. selfishness is *not* always harmful).

From this phenomenon, we argue that it is reasonable to introduce *leeches* into our model as in our preliminary assumptions, and actually there are lots of *leeches* existed in realistic systems. In other words, *what is real is rational and what is rational is real.*[1]

5.2. Download Bandwidth's Role

In the previous subsection, we have seen the download bandwidth's impact on the system performance. Intuitively, increasing the download bandwidth will lead to a shorter downloading time, as often observed in our daily experiences; but is this common sense always true? Now we study the impact of τ on the system performance (i.e. download bandwidth's role).

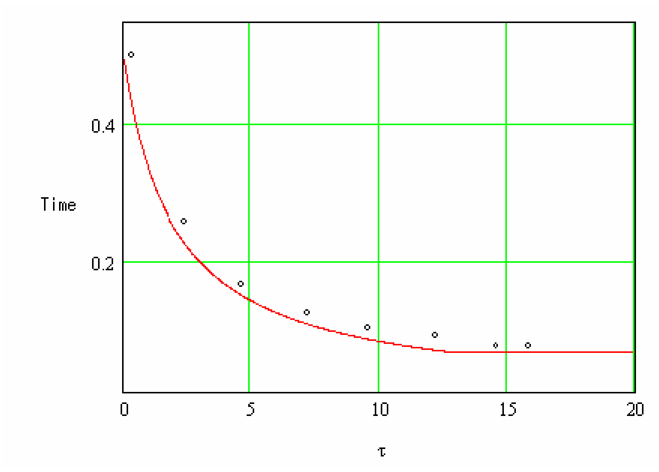


Figure 11. Impact of τ on Network Performance

The network parameters we have chosen are: $\beta = 0.2$, $\mu = 12$kbps, $\rho = 2$kbps, $\kappa = 50$kbps. Shown as the red curve in Figure 11, *Time* is a non-increasing function of τ. Besides, we can also derive the upper bound and lower bound of *Time* if we set $\tau = 0$ (i.e. the download channel is actually blocked) and $\tau = \infty$ (i.e. the download bandwidth is *much* higher than upload bandwidth) respectively.

The left half part of the curve justifies our intuition perfectly, but the right half seems to yaw from the common sense. The key to the phenomenon is still *bottleneck* bandwidth: initially, when τ increases, *Time* decreases accordingly because download bandwidth is the bottleneck now; however, once τ becomes big enough, increasing τ will not decrease *Time* any more, because the download bandwidth is no longer the bottleneck of the system performance.

In fact, if we consider the impact of μ on network performance (i.e. upload bandwidth's role), we will get a similar curve. From these phenomena, we argue that there are not always performance gains with increased download bandwidth, and the key to network performance gains is to keep a good balance of download bandwidth

[1] Taken from Hegel's famous dictum *Das Wirkliche sei vernuenftig und das Vernuenfitige wirklich.*

and upload bandwidth, and actually to increase bottleneck bandwidth. In other words, *every coin has two sides.*[2]

5.3. Importance of Seeds

The *seeds* are a special kind of peers, which upload but don't download. Compared to *leeches*, seeds can be deemed as *selfless* peers. Intuitively, it is very important to have seeds in the system; and in this subsection, we study the impact of κ on the system performance (i.e. seeds' contribution).

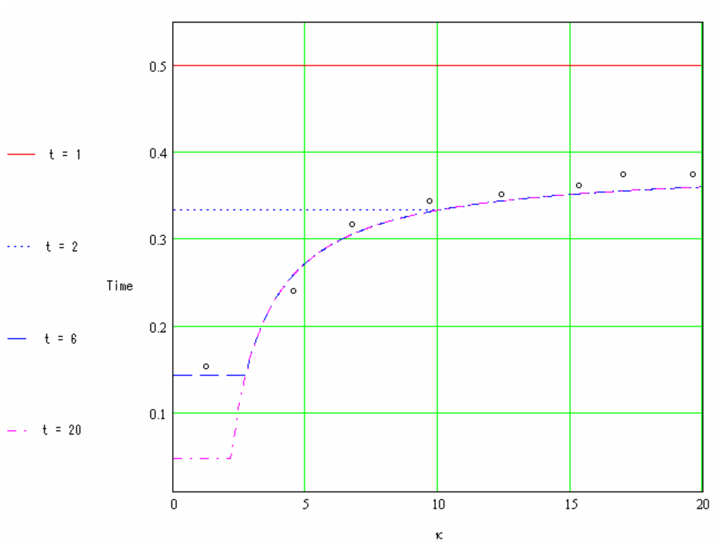


Figure 12. Impact of κ on Network Performance

The network parameters we have chosen are: β = 0.2, μ = 2kbps, ρ = 1kbps, τ_0 = 1kbps, τ_1 = 2kbps, τ_2 = 6kbps, τ_3 = 20kbps. With the curves shown in Figure 12, we are now not surprised to see the divisions of these curves and their singular points, because we already know their roots in the *bottleneck* bandwidth concept. Here we just briefly comment on the situation τ_2 = 6kbps because this speed seems to coincide with the practical speed of our daily cellular networks (e.g. GPRS): the ideal scenario is κ = 0 (i.e. all seeds are persistent in the network), where the lower bound of *Time* resides. As κ increases, initially, the slight loss of seeds doesn't degrade the system performance since the system is download bandwidth constrained; however, once κ is big enough, the system turns into upload bandwidth constrained, and the system performance degrades sharply with the loss of seeds; this also explains the singular point in the curve.

The realistic interpretation of *seeds* is service points or completed downloaders (but not all completed downloaders become seeds due to the existence of *leeches*), and the realistic meaning of the phenomenon is: it would be an effective way for mobile operators to improve QoS in such kind of systems via providing more service points.

[2] Ancient proverb.

6. Concluding Remarks

In this paper, we first studied the peer-to-peer systems over mobile ad hoc networks with a comparison of different settings for the peer-to-peer overlay and underlaying mobile ad hoc network. We show that cross-layer approach performs better than separating the overlay from the access networks. After characterizing the variability of the system by taking some preliminary assumptions, we then present a performance model which captures most facets of mobile peer-to-peer systems. We also briefly discussed three analytical examples on apply this model to capture the behavior of the system in steady states.

In order to make the paper concise, we didn't use the model to analyze the system in inequilibrious states, though it is not hard to simulate these cases with the given fluid model. We hope our results would potentially provide useful guidelines for mobile operators, value-added service providers and application developers to design and dimension mobile peer-to-peer systems, and as a foundation for our long term goals [51, 52].

References

[1] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, Z. Segall. When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks. In *Proc. 1st International Conference on Peer-to-Peer Computing (P2P 2001)*, Linkoping, Sweden, August 2001.

[2] M. Papadopouli and H. Schulzrinne. A Performance Analysis of 7DS a Peer-to-Peer Data Dissemination and Prefetching Tool for Mobile Users. In *Advances in wired and wireless communications, IEEE Sarnoff Symposium Digest*, 2001, Ewing, NJ.

[3] C. Lindemann and O. Waldhorst. A Distributed Search Service for Peer-to-Peer File Sharing in Mobile Applications. In *Proc. 2nd IEEE Conf. on Peer-to-Peer Computing (P2P 2002)*, 2002.

[4] A. Klemm, Ch. Lindemann, and O. Waldhorst. A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks. In *Proc. IEEE Vehicular Technology Conf.*, Orlando, FL, October 2003.

[5] S. K. Goel, M. Singh, D. Xu. Efficient Peer-to-Peer Data Dissemination in Mobile Ad-Hoc Networks. In *Proc. International Conference on Parallel Processing (ICPPW '02)*, IEEE Computer Society, 2002.

[6] G. Ding, B. Bhargava. Peer-to-peer File-sharing over Mobile Ad hoc Networks. In *Proc. 2nd IEEE Conf. on Pervasive Computing and Communications Workshops*. Orlando, Florida, 2004.

[7] H.Y. Hsieh and R. Sivakumar. On Using Peer-to-Peer Communication in Cellular Wireless Data Networks. In *IEEE Transaction on Mobile Computing*, vol. 3, no. 1, January-March 2004.

[8] B. Bakos, G. Csucs, L. Farkas, J. K. Nurminen. Peer-to-peer protocol evaluation in topologies resembling wireless networks. An Experiment with Gnutella Query Engine. In *Proc. International Conference on Networks*, Sydney, Oct., 2003.

[9] T. Hossfeld, K. Tutschku, F. U. Andersen, H. Meer, J. Oberender. Simulative Performance Evaluation of a Mobile Peer-to-Peer File-Sharing System. *Research Report 345*, University of Wurzburg, Nov. 2004.

[10] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, Z. Xu. Peer-to-Peer Computing. *Technical Report HPL-2002-57*, HP Labs.

[11] *MANET Implementation Survey*. Available at http://protean.itd.nrl.navy.mil/manet/survey/survey.html

[12] Gnutella: http://www.gnutella.com/

[13] Freenet: http://freenet.sourceforge.net/

[14] Kazaa: http://www.kazaa.com/

[15] eMule: http://www.emule-project.net/

[16] BitTorrent: http://bittorrent.com/

[17] *DSR IETF draft v1.0*. Available at http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt

[18] *AODV IETF draft v1.3*. Available at http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-13.txt

[19] Clip2. *The gnutella protocol specification v0.4 (document revision 1.2).* Available at http://www9.limewire.com/developer /gnutella protocol 0.4.pdf, Jun 2001.

[20] L. Yan and K. Sere. Stepwise Development of Peer-to-Peer Systems. In *Proc. 6th International Workshop in Formal Methods (IWFM'03).* Dublin, Ireland, July 2003.

[21] M. Ripeanu, I. Foster and A. Iamnitch. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. In *IEEE Internet Computing*, vol. 6(1) 2002.

[22] Y. Chawathe, S. Ratnasamy, L. Breslau, S. Shenker. Making Gnutella-like P2P Systems Scalable. In *Proceedings of ACM SIGCOMM,* 2003.

[23] D. B. Johnson, D. A. Maltz. Dynamic Source Routing in Ad-Hoc Wireless Networks. In *Mobile Computing*, Kluwer, 1996.

[24] C. E. Perkins and E. M. Royer. The Ad hoc On-Demand Distance Vector Protocol. In *Ad hoc Networking*. Addison-Wesley, 2000.

[25] F. Kojima, H. Harada and M. Fujise. A Study on Effective Packet Routing Scheme for Mobile Communication Network. In *Proc. 4th Intl. Symposium on Wireless Personal Multimedia Communications*, Denmark, Sept. 2001.

[26] L. Yan and J. Ni. Building a Formal Framework for Mobile Ad Hoc Computing. In *Proc. International Conf. on Computational Science (ICCS 2004).* Krakow, Poland, June 2004. LNCS 3036, Springer-Verlag.

[27] E. M. Royer and C. K. Toh. A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks. In *IEEE Personal Communications*, April 1999.

[28] Q. Lv, S. Ratnasamy and S. Shenker. Can Heterogeneity Make Gnutella Scalable? In *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, Cambridge, MA, March 2002.

[29] B. Yang and H. Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *Proc. Intl. Conf. on Distributed Systems (ICDCS)*, 2002.

[30] Y. Chawathe, S. Ratnasamy, L. Breslau, and S. Shenker. Making Gnutella-like P2P Systems Scalable. In *Proc. ACM SIGCOMM 2003*, Karlsruhe, Germany, August 2003.

[31] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM*, 2001.

[32] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Schenker. A scalable content-addressable network. In *Proc. Conf. on applications, technologies, architectures, and protocols for computer communications*, ACM, 2001.

[33] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, pages 329-350, November, 2001.

[34] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. In *IEEE Journal on Selected Areas in Communications*, January 2004, Vol. 22, No. 1.

[35] C. Plaxton, R. Rajaraman, A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. ACM SPAA*, Rhode Island, June 1997.

[36] S. Ratnasamy, S. Shenker, I. Stoica. Routing Algorithms for DHTs: Some Open Questions. In *Proc. 1st International Workshop on Peer-to-Peer Systems*, March 2002.

[37] H. Pucha, S. M. Das and Y. C. Hu. Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks. In *Proc. 6th IEEE Workshop on Mobile Computing Systems and Applications*, December 2004, UK.

[38] R. Schollmeier and I. Gruber. Routing in Peer-to-Peer and Mobile Ad Hoc Networks. A Comparison. In *Proc. International Workshop on Peer-to-Peer Computing*, Pisa, Italy, 2002.

[39] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *Proc. Middleware*, 2003.

[40] S. Corson and J. Macker. Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. *RFC 2501*, Jan. 1999.

[41] Z. Ge, D. Figueiredo, S. Jaiswal, J. F. Kurose, D. Towsley. Modeling peer-to-peer file sharing systems. In *Proc. IEEE Infocom*, 2003.

[42] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. 19th ACM symposium on Operating systems principles*, 2003.

[43] F. Clevenot and P. Nain. A Simple Fluid Model for the Analysis of the Squirrel Peer-to-Peer Caching System. In *Proc. IEEE Infocom*, 2004.

[44] X. Yang and G. Veciana. Service Capacity of Peer to Peer Networks. In *Proc. IEEE Infocom*, 2004.

[45] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In *Proc. ACM SIGCOMM*, 2004.

[46] F. L. Piccolo, G. Neglia. The Effect of Heterogeneous Link Capacities in BitTorrent-Like File Sharing Systems. In *Proc. Intl. Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P'04)*, Oct, 2004.

[47] J. Byers, M. Luby, M. Mitzenmacher. Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads. In *Proc. IEEE Infocom*, 1999.

[48] J. W. Byers, J. Considine, M. Mitzenmacher and S. Rost. Informed Content Delivery Across Adaptive Overlay Networks. In *Proc. ACM SIGCOMM*, 2002.

[49] S. Saroiu, P.K. Gummadi, S.D Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Proc. Multimedia Computing and Networking (MMCN'02)*, 2002.

[50] P. J. Denning and J. P. Buzen. The Operational Analysis of Queueing Network Models. In *ACM Computer Survey*, 1978.

[51] L. Yan, K. Sere, X. Zhou, and J. Pang. Towards an Integrated Architecture for Peer-to-Peer and Ad Hoc Overlay Network Applications. In *Proc. 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004)*, May 2004.

[52] L. Yan. MIN: Middleware for Network-Centric Ubiquitous Systems. In *IEEE Pervasive Computing*, Vol. 3, No. 3, 2004.

Paper VII

Formal Context-Aware Programming in Mobile Environments

L. Yan, K. Sere

Submitted to *Scientific Programming.*

Previous version available as: *L. Yan, K. Sere. A Formalism for Context-Aware Mobile Computing. In Proceedings of the 3rd International Symposium on Parallel and Distributed Computing and the 3rd International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04), July 2004. IEEE Computer Society Press.*

Formal Context-Aware Programming in Mobile Environments

Lu Yan, Kaisa Sere
Turku Centre for Computer Science (TUCS) and
Department of Computer Science, Åbo Akademi University,
FIN-20520 Turku, Finland.
{Lu.Yan, Kaisa.Sere}@abo.fi

Abstract

Mobile devices, such as mobile phones and PDAs, have gained wide-spread popularity. Applications for this kind of mobile devices have to adapt to changes in context, such as variations in network bandwidth, battery power, connectivity, reachability of services and hosts, and so on. In this paper, we define context-aware action systems that provides a systematic method for managing and processing context information. The meaning of context-aware action systems is defined in terms of classical action systems, so that the properties of context-aware action systems can be proved using standard action systems proof techniques. We describe the essential notions of this formalism and illustrate the framework with examples on context-aware services for mobile applications.

1 Introduction

Mobile computing devices, such as notebooks, mobile phones, PDAs and digital cameras have gained wide-spread popularity. Although these devices and their networking capabilities are becoming increasingly powerful, the design of mobile applications will continue to be constrained by physical limitations. Mobile devices will continue to be battery-dependent and users are reluctant to carry heavyweight devices. Networking capabilities will continue to be based on communication with basestations, with fluctuations in bandwidth depending on physical location. In order to provide acceptable QoS to the users, applications have to be *context-aware* [1] [2] [3] and able to adapt to context changes, such as variations in network bandwidth, exhaustion of battery power or reachability of services on other devices. This would require application developers, for example, to manage and process useful context information from the user's surroundings, and adapt to it accordingly. However, doing so would be extremely tedious and error-prone [4]. In order to ease the development of context-aware applications and ensure the correctness of their designs, we need new foundational ideas and principles.

Action systems [5], UNITY [23], and other similar state-based approach have proved to be effective in modeling and reasoning about distributed systems. In this paper, we extend the classical actions systems formalism with *context* information and define a novel *context-aware action systems* that provides a systematic method for managing and processing context information. The meaning of context-aware action systems is defined in terms of classical action systems, so that the properties of context-aware action systems can be proved using standard action systems proof techniques. Moreover, action systems are intended to be developed in a stepwise manner within an associated refinement calculus [6]. Hence, the development and reasoning about context-aware action systems can be carried out within this calculus ensuring the correctness of derived mobile applications [16].

Related works include Topological Action systems [15] which extends classical actions systems with a special topological variable *location*, Mobile UNITY [24] which is an extension of classical UNITY with a semantics based on temporal logic, and Ambient Calculus [25], a calculi approach dedicated to mobility in networks. Mobility aspect of distributed computing was extensively studied in those works, but no context-awareness was introduced. Recently, G. C. Roman extended the notion of Mobile UNITY and constructed a formal model of Context UNITY [26], which can be deemed as a simultaneous work related to our approach.

We proceed as follows. In Section 2, the action system formalism is extended with a notion of *context*. We describe some essential notions and properties of this formalism in Section 3. In Section 4, we illustrate the framework with examples on context-aware services for mobile applications. The concluding remarks are presented in Section 5.

Table 1. Abstract Syntax of Context

context	::=	*resourceList*
resourceList	::=	*resource resourceList* | ε
resource	::=	*rname oname valueList*
valueList	::=	*value valueList* | ε

Table 2. Formal Form of Context-Aware Action Systems

$$\begin{array}{lllll} \mathcal{A} & = & |[& \textbf{context} & c; \\ & & & \textbf{import} & i; \\ & & & \textbf{export} & e := e_0; \\ & & & \textbf{var} & v := v_0; \\ & & & \textbf{do}\ A\ \textbf{od} & \\ & &]| & & \end{array}$$

2 Context-Aware Action Systems

We define the notion *context* based on a collection of relations constraining when the computation can take place or where the data can reside. The abstract syntax of context is listed in Table 1, where *rname* $\in R$, being $R \subset \Sigma^*$, the set of all valid resource names over our alphabet Σ; *value* $\in V$, being V the set of all possible values of resources in R (e.g. IP address for hosts in reach, etc.); *oname* $\in O$, being O the set of all valid operator names that can be applied to values of monitorable resources (e.g. *equals, lessThan*).

We define the basic unit of execution in a context-aware mobile system to be a *context-aware action system*. Such a system comprises several sections and has the formal form as shown in Table 2. The first four sections are for context and variable declaration or use, while the last describes the computation involved in $\mathcal{A}$. Alternatively, we can write $\mathcal{A}$ in the explicit form as shown in Table 3.

The **context** section describes a set of *context* relations associated with $\mathcal{A}$. The abstract syntax of c is defined in Table 1. The content of this section can be optional. If not

Table 3. Explicit Form of Context-Aware Action Systems

$$\begin{array}{lllll} \mathcal{A} & = & |[& \textbf{import} & i; \\ & & & \textbf{export} & e := e_0; \\ & & & \textbf{var} & v := v_0; \\ & & & \textbf{do}\ A\ \textbf{od} & \\ & &]| & @c & \end{array}$$

Table 4. Definition of WP

$wp(abort, \mathcal{P})$	=	$false$
$wp(skip, \mathcal{P})$	=	$\mathcal{P}$
$wp(x := v, \mathcal{P})$	=	$\mathcal{P}[x/v]$
$wp(b \to A, \mathcal{P})$	=	$(b \Rightarrow wp(A, \mathcal{P}))$
$wp(A_1; A_2, \mathcal{P})$	=	$wp(A_1, wp(A_2, \mathcal{P}))$
$wp([]_I A_i, \mathcal{P})$	=	$\forall i \in I . wp(A_i, \mathcal{P})$
$wp(\text{if } b \text{ then } A_1$ else A_2 fi$, \mathcal{P})$	=	$(b \Rightarrow wp(A_1, \mathcal{P}) \wedge \neg b \Rightarrow wp(A_2, \mathcal{P}))$

specified, c is assigned to a default empty set ϕ, and the system degrades into a classical action system.

The **import** section describes the *imported* variables i that are not declared, but used in $\mathcal{A}$. The variables i are declared in some other context-aware action systems, and thus they model the communication between context-aware action systems.

The **export** section describes the *exported* variables e declared by $\mathcal{A}$. They can be used within $\mathcal{A}$ and also within other context-aware action systems that import them. Initially, they get the values e_0. If the initialization is missing, arbitrary values from the type sets of e are assigned as initial values.

The **var** section describes the *local* variables of context-aware action system $\mathcal{A}$. They can be used only within $\mathcal{A}$. Initially they are assigned values i_0, or, if the initialization is missing, some arbitrary values from their type sets.

Technically, all the used variables in **context**, **import**, and **export** sections are *global* variables, and only variables defined in **var** section are *local* ones.

The **do** $\cdots$ **od** section describes the *computation* involved in $\mathcal{A}$. An *action* is an atomic statement that can change the values of the local or global variables of the context-aware action system. An action A is defined by the following grammar:

$$\begin{array}{lll} A & ::= & abort | skip | x := v | \text{if } b \text{ then } A_1 \text{ else} A_2 \text{ fi} | \\ & & b \to A | A_1; A_2 | A_1 \ [] \ A_2 \end{array}$$

Here x is a list of attributes, v a list of values, b a predicate. Intuitively, *abort* is the action which always deadlocks, *skip* is a stuttering action, $x := v$ is a multiple assignment, $b \to A$ is a guarded action, $A_1; A_2$ is the sequential composition of two actions A_1 and A_2, if b then A_1 else A_2 fi is the conditional composition of two actions A_1 and A_2, and $A_1 \ [] \ A_2$ is the nondeterministic choice between two actions A_1 and A_2.

The semantics of an action A is described in terms of the weakest precondition predicate transformer, in the style of Dijkstra [7]. Given a predicate $\mathcal{P}$, the details of the definition of the function $wp(A, \mathcal{P})$ is listed in Table 4.

An important property of an action is its *enabledness*. The central part of this concept is the *guard condition*. We say that an action behaves miraculously when it establishes the postcondition *false*, which models an aborting state. Classically, the guard condition gA defined as $gA = \neg wp(A, false)$ gives those states in which an action behaves non-miraculously. In the context-aware action systems framework, we extend the the guard condition gA via incorporating *context* into the guard. The *guard* of the action A can now be defined as $gdA = c \wedge gA$, where c is the context and gA is the guard condition. An action A within a context-aware action system is said to be *enabled*, if its guard gdA evaluates to *true*. Action A can be chosen for execution only if it is enabled.

A context-aware action system is thus a set of actions operating on local and global variables. First, the variables are created and initialized. Then, repeatedly, enabled actions are non-deterministically chosen and executed. Actions operating on disjoint sets of variables can be executed in parallel. The computation terminates if no action is enabled, otherwise it continues infinitely. Actions are taken to be atomic, meaning that if an enabled action A is chosen for execution, then it is executed to completion without any interference from other actions of the system. This ensures that a parallel execution of a context-aware action system gives the same results as a sequential non-deterministic execution.

Compared to the classical action system approach, we can notice that the new thing here is the interpretation of *context*. Hence, we can say that context-aware action systems forms a subset of action systems. Intuitively, mobile computing can be modeled easily within our context-aware action systems framework, since *mobility* can always be treated as a special kind of context, i.e. *spatial context*. If we restrict the context part of our formalism into a set containing only *location* information, this formalism degrades into topological action systems [15], which is dedicated to mobile distributed computing. The similar phenomena can also be observed in Roman's paradigm of Context UNITY [26] and Mobile UNITY [24].

3 Essence of Context-Aware Action Systems

In this section we consider some essential notions and properties of context-aware action systems. The most important concepts in context-aware action systems are *parallel composition* and *prioritizing composition* [8].

3.1 Parallel composition

We have defined the context-aware action system as the basic unit of execution. In order to model a complex system, we still need a way to compose several context-aware action systems together. We first define the *parallel composition* of context-aware action systems. Consider the context-aware action systems $\mathcal{A}$ and $\mathcal{B}$ below:

$$\begin{array}{lll} \mathcal{A} = |[& \textbf{import} & i; \\ & \textbf{export} & e := e_0; \\ & \textbf{var} & v := v_0; \\ & \textbf{do}\ A\ \textbf{od} & \\ ]| & @c_1 & \end{array} \qquad \begin{array}{lll} \mathcal{B} = |[& \textbf{import} & j; \\ & \textbf{export} & f := f_0; \\ & \textbf{var} & w := w_0; \\ & \textbf{do}\ B\ \textbf{od} & \\ ]| & @c_2 & \end{array}$$

where the global and local variables declared in $\mathcal{A}$ and $\mathcal{B}$ are required to be distinct. We define the parallel composition $\mathcal{A}||\mathcal{B}$ of the context-aware action systems $\mathcal{A}$ and $\mathcal{B}$ to be the context-aware action system:

$$\begin{array}{ll} \mathcal{A}||\mathcal{B} = |[& \textbf{context}\ \ c_1, c_2; \\ & \textbf{import}\ \ k; \\ & \textbf{export}\ \ h := h_0; \\ & \textbf{var}\ \ u := u_0; \\ & \textbf{do}\ \ c_1 \rightarrow A \ [] \ c_2 \rightarrow B\ \ \textbf{od} \\ ]| & \end{array}$$

where $k = (i \cup j) - h$, $h = e \cup f$ and $u = v \cup w$. The initial values of the variables and the actions in $\mathcal{A}||\mathcal{B}$ consist of the initial variables and actions of the original action systems. The binary parallel composition operator $||$ is associative and commutative and thus extends naturally to the parallel composition of a finite set of context-aware action systems.

3.2 Prioritizing composition

We start by defining the prioritizing composition of actions, and then consider the prioritizing composition of action systems.

Let A, B, C be actions. The prioritizing composition $A//B$ selects the first operand if it is enabled, otherwise the second, the choice being deterministic.

$$A//B = A \ [] \ \neg gdA \rightarrow B$$

Since $A = gdA \rightarrow A$, the above definition can be equivalently stated as:

$$A//B = gdA \rightarrow A \ [] \ \neg gdA \rightarrow B$$

The prioritizing composition of two actions is enabled if either operand is:

$$gd(A//B) = gdA \vee gdB$$

Prioritizing composition of actions is associative, allowing parentheses to be omitted in repeated applications.

$$(A//B)//C = A//(B//C)$$

Prioritizing composition of actions distributes over choice to the right, but does not distribute over choice to the left in general.

$$A//(B \,[]\, C) = (A//B) \,[]\, (A//C)$$

Let $\mathcal{A}$ and $\mathcal{B}$ be context-aware action systems given below:

$$\begin{array}{llll} \mathcal{A} = |[& \textbf{import} & i; \\ & \textbf{export} & e := e_0; \\ & \textbf{var} & v := v_0; \\ & \textbf{do}\ A\ \textbf{od} & \\ ]| & @c_1 & \end{array} \qquad \begin{array}{llll} \mathcal{B} = |[& \textbf{import} & j; \\ & \textbf{export} & f := f_0; \\ & \textbf{var} & w := w_0; \\ & \textbf{do}\ B\ \textbf{od} & \\ ]| & @c_2 & \end{array}$$

where the global and local variables declared in $\mathcal{A}$ and $\mathcal{B}$ are required to be distinct. The prioritizing composition $\mathcal{A}//\mathcal{B}$ combines $\mathcal{A}$ and $\mathcal{B}$ in a way that preference is given to the action of $\mathcal{A}$. The choice between the action of $\mathcal{A}$ and $\mathcal{B}$ is deterministic in the sense that when both are enabled, the action of $\mathcal{A}$ is taken.

$$\begin{array}{lll} \mathcal{A}//\mathcal{B} = |[& \textbf{context} & c_1, c_2; \\ & \textbf{import} & k; \\ & \textbf{export} & h := h_0; \\ & \textbf{var} & u := u_0; \\ & \textbf{do} & c_1 \to A \ //\ c_2 \to B \quad \textbf{od} \\ ]| & & \end{array}$$

where $k = (i \cup j) - h$, $h = e \cup f$ and $u = v \cup w$. The initial values of the variables and the actions in $\mathcal{A}||\mathcal{B}$ consist of the initial variables and actions of the original action systems.

Prioritizing composition of context-aware action systems is associative, allowing us to omit parentheses in repeated application.

$$(\mathcal{A}//\mathcal{B})//C = \mathcal{A}//(\mathcal{B}//C)$$

Let $\mathcal{G}$ be a context-aware action system without local variables, i.e. of the form

$$\begin{array}{lll} \mathcal{G} = |[& \textbf{import} & i; \\ & \textbf{export} & e := e_0; \\ & \textbf{do}\ G\ \textbf{od} & \\ ]| & @c & \end{array}$$

Prioritizing composition with a context-aware action system without local variables distributes over parallel composition to the right, but does not distribute over parallel composition to the left in general:

$$\mathcal{G}//(\mathcal{A}||\mathcal{B}) = (\mathcal{G}//\mathcal{A})||(\mathcal{G}//\mathcal{B})$$

3.3 Nesting

The number of entities that share a resource might change within a complex system. This feather is modeled for context-aware action systems by hiding (or revealing) exported (or local) variables. We define the *nesting* $|[\mathcal{A}]|$ of the context-aware action system $\mathcal{A}$ as follows:

$$\begin{array}{llll} |[\mathcal{A}]| = & |[\ |[& \textbf{import} & z; \\ & & \textbf{export} & y := y_0; \\ & & \textbf{var} & x := x_0; \\ & & \textbf{do}\ B\ \textbf{od} \quad]|\ @a\ ]| & \\ = & |[& \textbf{import} & z; \\ & & \textbf{var} & y, x := y_0, x_0; \\ & & \textbf{do}\ C\ \textbf{od} \quad]|\ @a & \end{array}$$

The exported variable y in $\mathcal{A}$ is a local variable in the nested action system $|[\mathcal{A}]|$. Therefore, y is provided as an exported variable only to a certain domain and hidden from other domains. Some security means can be modeled using this feature.

4 Context-Aware Mobile Computing

Using the framework of context-aware action systems, we can now model mobile computing in an extremely dynamic context: location changes all the time while moving around with our portable devices, and so the services and devices in reach; local resource availability varies quickly as well, such as memory availability, bandwidth and battery power. In order to maintain reasonable QoS to the users, applications have to be context-aware. In this section, we examine a representative set of context-aware services found in the literature, abstract their key features, and suggest ways to model them in the context-aware action systems framework.

4.1 Conference assistant example

Initial work in context-aware computing resulted in the development of applications that can use context definitions to support everyday behaviors, such as Active Badge [9] and PARCTab [10]. Another kind of typical context-aware applications relate to the development of guides, e.g. Cyberguide [11] and GUIDE [12]. Therefore, we present a context-aware scenario similar to [13] and [14] as an example to show how this context-aware action systems framework can be effectively used to model context-aware services for mobile applications.

Imagine that Kaisa is attending a conference with her own Smart Phone. When arriving at the conference location, she is provided with a mobile application to be installed on her own portable device that, based on a wireless

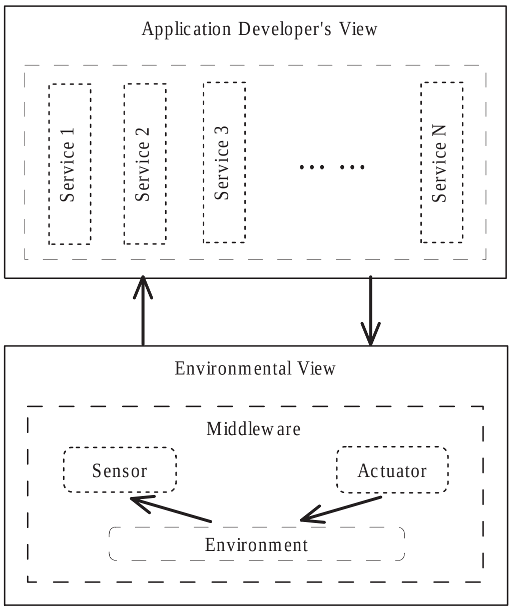


Figure 1. The Overall Infrastructure

network infrastructure, allow attendees to access the proceedings online, browse through the technical program, select the presentation they wish to attend and exchange messages with other attendees. These services may have to be delivered in different ways when requested in different contexts, in order to meet the users' needs.

4.1.1 Modeling the environment

The ideas behind this scenario are rooted in the notion that mobile application development could be simplified if the retrieval and maintenance of context information were to be delegated to the software support infrastructure without loss of flexibility and generality as shown in Figure 1.

To ease the prototyping of a context aware application, we proposed a middleware for network-centric ubiquitous systems in [22] from which an application developer can derive specific services. This layer takes care of most low-level context-aware functions: collecting sensor data, combining data from multiple sensors, translating sensor data into alternate formats, and contains the infrastructure required for distributed peer-to-peer storage, communication via XML over HTTP, and software event monitoring.

Here we introduce a simplified model of the environment in this paper, which just retrieves and maintains necessary environment variables, i.e.

$$\mathcal{M}_{iddleware} = \mathcal{S}_{ensor} \| \mathcal{A}_{ctuator}$$

where Sensor and Actuator, modeled via classical action systems in Table 5, make the context information accessible

Table 5. Sensor and Actuator

```
$\mathcal{S}_{ensor}$ = |[  **import**  $read$;
                   **export**  $value$;
                   **var**     $env$;
                   **do**
                             $read = true \rightarrow value = env;$
                                           $read = false$
                   **od**
              ]|

$\mathcal{A}_{ctuator}$ = |[  **import**  $write, value$;
                     **var**     $env$;
                     **do**
                               $write = true \rightarrow env = value;$
                                              $write = false$
                     **od**
                ]|
```

and updated from the application developer's view; more sophisticated models can be found in [21]. The benefit of this approach is obvious: since context information actually belongs to the environmental view, given proper middleware, application developers will usually focus on service construct and not necessarily care too much about the low-level context-aware operations; thus it will ease application development by taking advantage of this abstraction.

4.1.2 Reminding service

In the remaining part of the paper, we focus on an application developer's view and more details are given on service derivation within our *context-aware action systems* framework.

Let us consider a reminding service that alerts an attendee of the coming presentation to attend 5 minutes before it starts. Based on the functionality of Kaisa's Smart Phone, the following profiles can be used to remind her of coming events: *sound alert*, particularly useful to capture user attention in noisy and open air place; and *vibra alert*, to capture user attention without disturbing anyone else (e.g. when attending a talk).

We model the reminding service in context-aware action systems as follows. The reminding service $\mathcal{R}_{eminder}$ is a parallel composition of sound alter service $\mathcal{A}_{sound}$ and vibra alert service $\mathcal{A}_{vibra}$ in Table 6, i.e.

$$\mathcal{R}_{eminder} = \mathcal{A}_{sound} \| \mathcal{A}_{vibra}$$

We model the current time and the coming presentation as imported variables *now* and *talk*, and current status of our service as an export variable *alert*. Then, we store our preference in the variable *schedule* for reference. The context-

Table 6. Alert Services

```
A_sound = |[ imp   $now, talk;$
             exp   $alert;$
             var   $schedule;$
             do
                   $schedule.time - now > 5$
                   $\vee now > schedule.time$
                   $\rightarrow alert := off$
                   $[]\ 0 < schedule.time - now < 5$
                     $\rightarrow$ if $schedule.title = talk$
                        $\rightarrow alert := sound$ fi
             od
          ]| @outdoor

A_vibra = |[ imp   $now, talk;$
             exp   $alert;$
             var   $schedule;$
             do
                   $schedule.time - now > 5$
                   $\vee now > schedule.time$
                   $\rightarrow alert := off$
                   $[]\ 0 < schedule.time - now < 5$
                     $\rightarrow$ if $schedule.title = talk$
                        $\rightarrow alert := vibra$ fi
             od
          ]| @conference
```

aware action system formalism has context information c to constrain at what situation a service is delivered. For the reminding service we define, there are two sorts of context: *outdoor* profile and *conference* profile, where

$$\begin{aligned} outdoor &= \{location = outside\} \\ conference &= \{location = conferenceRoom\} \end{aligned}$$

Using the properties presented in the previous section and the techniques discussed in [15] and [16], we can transform these context-aware action systems into one action system and further refine it within its associated refinement calculus [6]. A possible result is shown in Table 7.

With the above refined specification, we can derive a design pattern [19] as shown in Figure 2 to ease the software development for *reminding service*: as an attendee moves around the conference places, his or her context variable, *location*, defined to contain the current location information, changes in response to the available context. If the new context matches some particular locations, the attendee's reminding policy is updated to adapt the application to the new environment.

Table 7. Reminding Service

```
R_eminder = |[ context  $outdoor, conference;$
               imp      $now, talk, location;$
               exp      $alert;$
               var      $schedule;$
               do
                        $schedule.time - now > 5$
                        $\vee now > schedule.time$
                        $\rightarrow alert := off$
                        $[]\ 0 < schedule.time - now < 5$
                          $\rightarrow$ if $schedule.title = talk$
                             $\rightarrow$ if $location = outside$
                                $\rightarrow alert := sound$
                             $[]\ location = conferenceRoom$
                                $\rightarrow alert := vibra$ fi
                             fi
               od
            ]|
```

4.1.3 Messaging service

The messaging service enables an attendee to exchange messages with other attendees. Attendees can exchange messages using the following profiles: *SMS*, to exchange messages in plain text; *MMS*, to send messages comprising a combination of text, sounds, images and video; *EMS*, to send encrypted messages. The messaging service is an example of peer-to-peer service, where any number of peers may participate in the delivery of the service.

Let us assume, for example, another attendee Lu is willing to exchange messages with Kaisa, but he is using a PDA, which has a different profiling policy than the Smart Phone. We model this with two context-aware action systems:

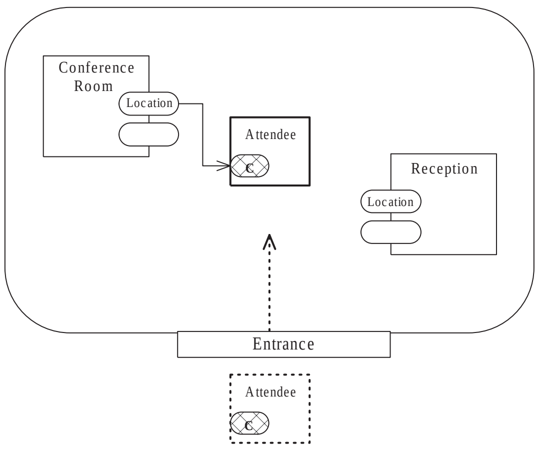


Figure 2. Reminding Service

Table 8. Contexts for Smart Phone and PDA

Profile	Context
SMS_{phone}	
MMS_{phone}	$bandwidth > 70\%$
	$battery > 50\%$
EMS_{phone}	$battery > 20\%$
SMS_{PDA}	$bandwidth > 5\%$
MMS_{PDA}	$bandwidth > 40\%$
	$battery > 25\%$
EMS_{PDA}	$bandwidth > 75\%$

$$\begin{aligned}\mathcal{K}_{aisa} &= SMS_{phone}||MMS_{phone}||EMS_{phone}\\ \mathcal{L}_{u} &= SMS_{PDA}||MMS_{PDA}||EMS_{PDA}\end{aligned}$$

where the contexts are defined in Table 8. Note that no context information is associated to SMS_{phone} context of Kaisa's profile: this means that this action is always available, regardless of current context. At any time, attendees may change their preferences through the user interface that the conference application provides; the application, in turn, dynamically updates the context information encoded in their profiles, in order to take the new preference into account.

The interesting part of the messaging service is the specification of *context* itself. Let us consider, for example, the messaging service is requested when Kaisa's bandwidth is greater than 70% and battery availability is greater than 50%, all three profiles: SMS_{phone}, MMS_{phone} and EMS_{phone} can be applied. In this sample, the messaging service will be delivered via *SMS*, *MMS* and *EMS* simultaneously, which is unnecessary and should be avoid in the real world.

In case we need to ensure that a service is delivered via *only* one context-aware action system even if several different context-aware action systems can be used, a *conflict* [4] may arise due to the different contexts themselves or due to changes in context. There has been some research on *conflict resolution* and several schemes are proposed. A critical literature review in this area can be found in [17]. In the context-aware action system framework, we implement a priority assignment scheme [18] for conflict resolution, where the order of prioritizing composition reflects the user's preferences.

$$\begin{aligned}\mathcal{K}_{aisa} &= MMS_{phone}//EMS_{phone}//SMS_{phone}\\ \mathcal{L}_{u} &= EMS_{PDA}//MMS_{PDA}//SMS_{PDA}\end{aligned}$$

4.2 The whole system

Let us imagine that, at the moment, the attendee Lu opens his Tablet PC to enable better communication with Kaisa. Obviously, this kind of mobile devices has a different profiling policy than the previous two. We model the situation with a new context-aware action system $\mathcal{L}_u'$:

Table 9. Contexts for Tablet PC

Profile	Context
SMS_{tab}	
MMS_{tab}	$bandwidth > 25\%$
	$battery > 10\%$
EMS_{tab}	$battery > 5\%$

$$\begin{aligned}\mathcal{L}_u' &= \mathcal{L}_{u1}||\mathcal{L}_{u2}\\ \mathcal{L}_{u1} &= EMS_{PDA}//MMS_{PDA}//SMS_{PDA}\\ \mathcal{L}_{u2} &= MMS_{tab}//EMS_{tab}//SMS_{tab}\end{aligned}$$

where the contexts are defined in Table 9. As the result, the *messaging service* is now modeled as follows:

$$\mathcal{K}_{aisa}||\mathcal{L}_u' = \mathcal{K}_{aisa}||(\mathcal{L}_{u1}||\mathcal{L}_{u2})$$

The final application is a parallel composition of all involved service providers:

$$\begin{aligned}&\mathcal{S}_{ervices}\\ &= \mathcal{R}_{eminder}||(\mathcal{K}_{aisa}||\mathcal{L}_u')\\ &= (\mathcal{A}_{sound}||\mathcal{A}_{vibra})||(\mathcal{K}_{aisa}||(\mathcal{L}_{u1}||\mathcal{L}_{u2})\\ &= \mathcal{A}_{sound}||\mathcal{A}_{vibra}||(MMS_{phone}//EMS_{phone}//SMS_{phone})\\ &\quad ||(EMS_{PDA}//MMS_{PDA}//SMS_{PDA})\\ &\quad ||(MMS_{tab}//EMS_{tab}//SMS_{tab})\end{aligned}$$

and the whole system is modeled as the interaction between the application view and the environment view:

$$\mathcal{S}_{ystem} = \mathcal{S}_{ervices}||\mathcal{M}_{iddleware}$$

where *services* leads to the final program to be deployed into attendee's mobile devices, and *middleware* is the supporting software existed in attendee's mobile devices and preinstalled at the conference venue.

5 Concluding Remarks

The increasing popularity of portable devices and recent advances in wireless network technologies are facilitating the engineering of new classes of distributed systems, which present challenging problems to designers. To harness the flexibility and power of these rapidly evolving, network and mobile computing systems, and in particular, to meet the need for context-awareness and adaptation, we need to come up with new foundational ideas and effective principles for building and analyzing such systems.

The novel contribution of this paper is the formal design and formalism that facilitate the development of context-aware applications. In particular, we have described a formal approach to context-aware mobile computing: we offer the *context-aware action systems* framework, which provides a systematic method for managing and processing context information, defined on a subset of the classical action systems. Besides the essential notions and properties of this formalism, we demonstrate how this formalism can effectively be used to model context-aware services for mobile applications with examples.

Future improvements and extensions of the context-aware action systems framework span towards different directions. *Conflict resolution* has been a very active research field in context-aware mobile computing. In our paper, we implement a static conflict resolution scheme like [18] within the context-aware action systems framework, i.e. it is up to the user's preferences to decide the way of conflict resolution. As a future work, we plan to introduce more dynamic schemes like [20] to the context-aware action system framework towards a better utilization of context information.

References

[1] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications.* Santa Cruz, CA, Dec. 1994.

[2] P.J. Brown, J.D. Bovey, and X. Chen. Context-Aware Applications: from the Laboratory to the Marketplace. In *IEEE Personal Communications.* 4(5): 58-64, 1997.

[3] A.K. Dey and G.D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness.* The Hague, Netherlands, April 2000.

[4] L. Capra, W. Emmerich, and C. Mascolo. CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications. In *IEEE Transactions on Software Engineering.* October 2003. (Vol. 29, No. 10).

[5] R.J.R. Back and K. Sere. From Action Systems to Modular Systems. In *Software - Concepts and Tools.* (1996) 17: 26-39.

[6] R.J. Back and J. Wright. *Refinement Calculus: A Systematic Introduction.* Graduate Texts in Computer Science, Springer-Verlag, 1998.

[7] E.W. Dijkstra: *A Discipline of Programming.* Prentice-Hall International, 1976.

[8] E. Sekerinski and K. Sere. A Theory of Prioritizing Composition. In *The Computer Journal.* Vol. 39, No. 8, 1996.

[9] A. Harter and A. Hopper. A distributed location system for the active office. In *IEEE Networks.* (1994) 8: 62–70.

[10] R. Want. An overview of the PARCTab ubiquitous computing environment. In *IEEE Personal Communications.* (1995) 2: 28–33.

[11] G. Abowd, C. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: A mobile context-aware tour guide. In *ACM Wireless Networks.* (1997) 3: 421–433.

[12] K. Cheverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Experiences of developing and deploying a context-aware tourist guide: The GUIDE project. In *Proceedings of the 6th annual international conference on Mobile computing and networking.* Boston, MA, 2000.

[13] A.K. Day, D. Salber, G.D. Abowd, and M. Futakawa. The Conference Assistant: Combining Context-Awareness with Wearable Computing. In *Proceedings of the 3rd International Symposium on Wearable Computers.* San Francisco, CA, Oct. 1999.

[14] A. Asthana, M. Cravatts, and P. Kryzyzanowski. An Indoor Wireless System for Personalized Shopping Assistance. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications.* Santa Cruz, CA, Dec. 1994.

[15] L. Petre, K. Sere, and M. Waldén. A Topological Approach to Distributed Computing. In *Proceedings of WDS'99 - Workshop on Distributed Systems.* Iasi, Romania, Sept. 1999. ENTCS 28, Elsevier Science.

[16] K. Sere and M. Waldén. Data Refinement of Remote Procedures. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Software (TACS97).* Sendai, Japan, Sept. 1997. LNCS 1281, Springer-Verlag.

[17] A.V. Lamsweerde, R. Darimont, and E. Letier. Managing Conflicts in Goal-Driven Requirements Engineering. In *IEEE Transactions on Software Engineering.* Vol. 24, No. 11, Nov. 1998.

[18] R.M. Sivasankaran, J.A. Stankovic, D. Towsley, B. Purimetla, and K. Ramamritham. Priority Assignment in Real-Time Active Databases. In *The International Journal on Very Large Data Bases.* Vol. 5, No. 1, January 1996.

[19] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, A System of Patterns.* John wiley & Sons, 1996.

[20] L. Capra, W. Emmerich, and C. Mascolo. A Micro-Economic Approach to Conflict Resolution in Mobile Computing. In *Proceedings of the 10th International Symposium on the Foundations of Software Engineering (FSE-10)*. Charleston, South Carolina, USA, Nov. 2002.

[21] L. Yan, K. Sere, X. Zhou, and J. Pang. Towards an Integrated Architecture for Peer-to-Peer and Ad Hoc Overlay Network Applications. In *Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004)*. Suzhou, China, May 2004.

[22] L. Yan. MIN: Middleware for Network-Centric Ubiquitous Systems. In *IEEE Pervasive Computing*, Vol. 3, No. 3, July - September 2004.

[23] K. Chandy and J. Misra. *Paralle Program Design: A Foundation*. Addison-Wesley, 1998.

[24] G. C. Roman and P. J. McCann. A Notation and Logic for Mobile Computing. In *Formal Methods in System Design*, Vol. 20, No. 1, 2002, pp. 47-68.

[25] L. Cardelli and A. D. Gordon. Mobile Ambients. In *Foundations of Software Science and Computational Structures*. LNCS 1378, pp. 140-155, Springer-Verlag, 1998.

[26] G. C. Roman, C. Julien and J. Payton. A Formal Treatment of Context-Awareness. In *Proceedings of the 7th Fundamental Approaches to Software Engineering Conference (FASE 2004)*. Barcelona, Spain, April 2004. LNCS 2984, Springer-Verlag.

Paper VIII

Implementing an Asynchronous Java Accelerator for Ubiquitous Computing

Z. Liang, L. Yan, J. Plosila, K. Sere

Submitted to *Journal of Embedded Computing*.

Previous version available as: *L. Yan, Z. Liang. Accelerating Java for Ubiquitous Devices. In Proceedings of the 4th International Conference on Computer and Information Technology (CIT'04), September 2004. IEEE Computer Society Press.*

Implementing an Asynchronous Java Accelerator for Ubiquitous Computing

Zheng Liang, Lu Yan, Juha Plosila, Kaisa Sere *Member, IEEE*

***Abstract*—Java is ideal for embedded and network computing applications. In this paper, we propose a hardware accelerated JVM with an asynchronous java accelerator, which can be integrated with most existing processors and run time operation systems. The architecture of the java accelerator was specially designed for low power consumption: 1. The chip is designed in asynchronous style and no clock is needed. 2. A novel branch prediction unit and decoded bytecode caches are integrated to eliminate the need of external memory access to the least. 3. The instruction folding unit specially designed for java bytecodes can effectively improve performance and decrease power consumption. Finally, hardware/software co-simulation with SystemC is discussed.**

***Keywords*—Asynchronous circuit design, System level modeling, Low power design, JVM, Java bytecode.**

I. Introduction

JAVA is the most popular and portable languages for its " write once, run any where" promise. It is expected that this enabling technology will make it much easier to develop portable software and standardized interfaces that span a spectrum of hardware platforms.

Java applications are first compiled into bytecode streams to execute in the Java Virtual Machine (JVM). Bytecode representations are portable formats that allow programs, whether small applets in embedded systems or large desktop applications, to run on many platforms. The core of the JVM implementation is the execution engine that executes bytecode instructions. It is important that the JVM provides an efficient execution/runtime environment across diverse hardware platforms.

A significant disadvantage of Java applications in embedded systems is the low performance. The software mode execution engine is quite slow in interpreter or bigger code size in Just-in-Time (JIT) compiler [1]. Silicon implementation can be optimized to deliver much better performance than software mode.

The power consumption in embedded systems comes from two sources: processor instruction operation and memory access. Chip architecture and fabrication technology are key to limiting power consumption.

In this paper, we propose a scheme of hardware accelerated JVM for existing embedded systems. The accelerator is designed for low power consumption, and can be integrated into most existing processors and run time operation systems (RTOS). To meet the low power constraint, the accelerator is completely designed in asynchronous style.

The remainder of this paper is organized as follows. We introduce the internal architecture of the JVM and system workflow with Java accelerator in Section 2. Asynchronous circuit design style, as well as the benefits, is introduced in Section 3. The architecture of our asynchronous Java accelerator is presented in Section 4. In section 5, the system level simulation and system power estimation are stated. We conclude the paper in Section 6.

II. JVM System Framework

Each Java application runs inside its own Java virtual machine. In the Java virtual machine specification, the behavior of a virtual machine instance is described in terms of subsystems, memory areas, data types and instructions. These components describe an abstract inner architecture of the abstract Java virtual machine. As shown in Fig.1, each Java virtual machine has a class loader subsystem, which is a mechanism for loading types (classes and interfaces) when given fully qualified names. Each JVM also has an execution engine, which is a mechanism responsible for executing the instructions contained in the methods of loaded classes. The Java virtual machine organizes the memory it needs to execute a program into several runtime data areas.

Each thread of a running Java application is a distinct instance of the virtual machine's execution engine. In a thread, bytecode execution can be implemented in either software or hardware [2]:

--Interpreter: it's just like a software emulation of the virtual machine, in this case, Java interpreter has an additional overhead and more executing cycles than just the bytecodes.

--Just-in-Time (JIT) compiler: it compiles a Java method into native instructions on the fly and caches the native sequence. The JIT compilers' memory requirement is pretty high for embedded systems and pervasive computing applications.

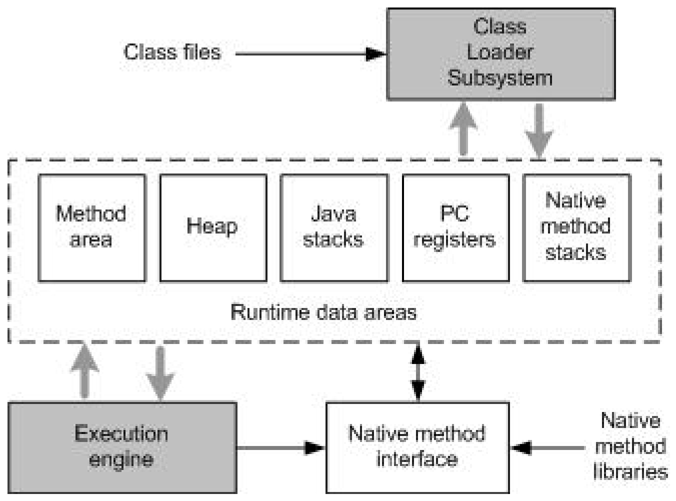


Fig. 1. Internal Architecture of JVM

--Hardware accelerator: one is to translate the bytecode into binary machine code of native main processor; another mode is to work as coprocessor which specially executes the bytecode instruction.

-- Java native processor: implements the JVM directly on silicon. It not only avoids the overhead of translation of the bytecodes to another processor's native language, but also provides support for Java runtime features.

Since our target is for most existing processors, coprocessor mode is the best choice, which means frequently used bytecode instructions are to be executed by the java accelerator.

Our java accelerator is designed for low end embedded equipments, where only one CPU accesses memory and works as master module in system bus.

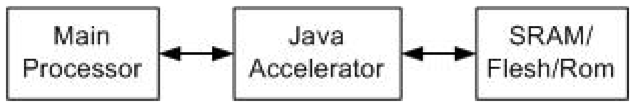


Fig. 2. Connection for Java Accelerator

As shown in Fig. 2, during program running, java accelerator is a filter between main processor and external memory. If the main processor are executing non-java task, or the address of accessed memory is out of current java stack, the java accelerator will be transparent.

When the java accelerator works with the main processor, the processor needs to configure the java accelerator in the beginning of every java thread. (i.e. set new PC, stack address, segment offset, etc in the java accelerator) Such kind of information storing will consume some I/O space of the processor. After the beginning of a java bytecode stream, the processor halts and the java accelerator will fetch bytecode instructions and data from memory. As shown in Fig.3, when the java accelerator encounters trap instruction which the accelerator can't implement, it sends interrupt and moves its control to the processor, and the processor will access the stack cache in the java accelerator as I/O operations.

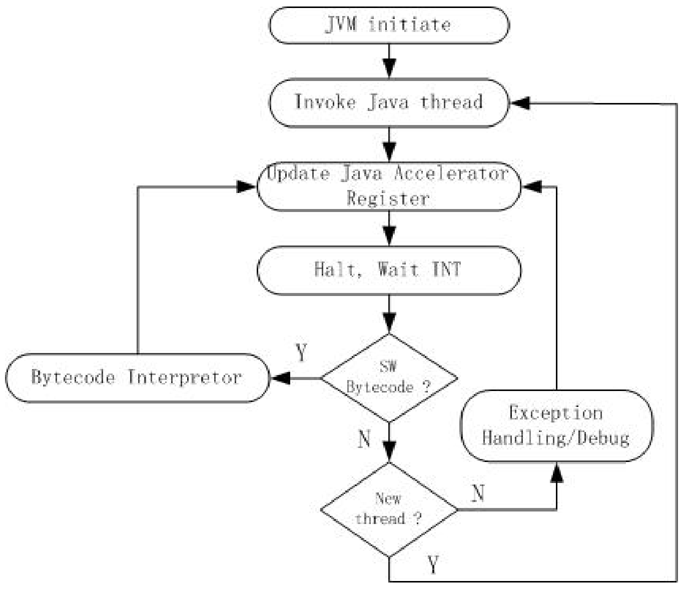


Fig. 3. JVM Workflow

In bytecode trap handling, the main processor takes operands from java stack, executes subroutines for trapped java bytecodes, and writes results back to java stack top. In case that the next bytecode should be executed in hardware, the main processor halts and updates PC in the java accelerator to invoke it. In this way, the java accelerator will be able to continue to use the new result in stack. With this scheme, our java accelerate can be integrated into most existing processors, and no bus arbiter is needed.

III. Asynchronous Design

Three main areas may benefit from an asynchronous design style: global synchronization, performance and power consumption.

A. Global synchronization

With the increase of the degree of on-chip integration, it is becoming increasingly difficult to maintain the global synchronization required in a clocked system. The difficulty lies in distributing the clock signal across the silicon in such a way that all elements receive a transition of the clock at the same time. If the clock skew is large, the clock period must be extended to ensure correct operation, and as a result, the maximum frequency is limited by the on-chip skew. Since asynchronous circuits have no global clock, there is no such constraint to satisfy and the complex clock driver network is not required.

The java accelerator is a small chip with limited area. The clock skew, if designed in synchronous, is not so obvious to baffle performance improving. But if we integrate the accelerator core into the SoC platform, the synchronization problem still remains. Furthermore, the java bytecode instructions are in variable length. It's more convenient to design the control logic with asynchronous circuits.

B. *Performance*

Normal synchronous design is optimized for worst-case conditions. The minimum clock period (and hence maximum frequency) is constrained by the operation that takes the longest time to complete.

The speed of a particular operation is affected by a number of independent factors:

--Variation in silicon processing of CMOS circuits leads to variation in transistor strengths between limits.

--Logic functions may have certain input data values that require more time to evaluate than the average case.

--The power supply voltage and temperature of a CMOS circuit affects its speed.

Within an asynchronous system it is possible to construct circuits optimized for the typical case; worst-case operations usually take longer time.

C. *Power consumption*

In CMOS technology, the power dissipated is proportional to the frequency of the clock, and the clock line tree itself is a heavy load, requiring large drivers. Decreasing the power supply can reduces the power, but there are limits to how low the supply voltage can go before the device stops functioning correctly. In an asynchronous system without any clock, actually only the required function module works, so it doesn't dissipate any power in modules that are not required.

The other advantage of asynchronous system is its EMC ability and Modularity. There are also disadvantages of the asynchronous design can't be ignored.

In a synchronous system, every processing stage must complete its activity in less than the duration of the clock period. An asynchronous system requires extra hardware to allow each block to perform local synchronizations to pass data to other blocks. Furthermore, to exploit data-dependent evaluation times, extra completion detection logic is needed. It adds complexity that results to larger circuits and more difficult design process.

Verification is also difficult due to the non-deterministic behavior of arbiter element, and deadlock is not easy to detect without exhaustive state space exploration. Testing for fabrication faults in asynchronous systems is another major obstacle due to the nondeterministic behavior of arbiter elements.

As an accelerator, the java accelerator is a small chip. Although maybe the power consumption of clock tree is not a big part comparing with other module, the asynchronous style is quite convenient for the micro-pipeline design in the java accelerator.

In our java accelerator, 4-phase handshake protocol was applied. Generally, 4-phase protocol control circuits are often simpler than those of the equivalent 2-phase systems. The signaling lines can be used to drive level-controlled latches and the like directly. The single-rail encoding was applied for data representation in the java accelerator, which is simpler than any other data presentation scheme. The die area requirements are similar to those of asynchronous designs, so any arithmetic components constructed for reuse in asynchronous systems can be used in this scheme.

Thus, the java accelerator chip is designed in asynchronous style for the strict power limitation in embedded systems.

Manual designing any complex asynchronous system is difficult. Balsa [3] is both a framework for synthesizing asynchronous (clockless) hardware systems and the language for describing such systems. The advantage of this approach is that the compilation is transparent: there is a one-to-one mapping between the language constructs in the specification and the intermediate handshake circuits that are produced.

The whole accelerator was first described in the Balsa language. With the toolkit provided by Manchester University, synthesis and simulation were done. But the completely synthesized chip has unoptimized performance and die area, so the final gate level design was carried out with hand drawing combining the Balsa synthesis tool. The silicon design based on UMC $0.13\mu m$ technology of the chip is underway.

IV. Accelerator Architecture

In this section, the function and architecture of our embedded java accelerator are specified.

The java accelerator is the only chip manufactured in this work. Our java accelerator is an independent coprocessor to facilitate java application running. It can be integrated easily with most existing 32/16 bits embedded processors and operation systems. About 130 bytecode instructions are implemented in hardware, and the rest are done in software with interpreter mode.

There are totally 245 java bytecode instructions, where standard bytecode opcodes ranging from 0 to 201, quick bytecodes ranging from 203 to 245. In this chip, we only consider hardware implementation of standard bytecode instructions. The quick bytecodes can be implemented by software interpreter.

At bytecode level, 45 out of 255 bytecodes constitute 90 percent of most dynamic bytecode streams. Except when using smaller block sizes for data caches or using branch predictors specially tailored for indirect branches, optimizing caches and branch predictors will not have a major impact on performance of interpreted java execution [1].

For standard bytecodes, integer division and remainder, all the float point arithmetic instructions are implemented in software. For long operands, "mul", "div" and "rem" are done in software. Most array instructions are also done in software. Such scheme can guarantee the control signals are hard wired, so only simple control logic is needed.

The chip frame is shown in Fig. 4. To meet the requirement of most low end microcontrollers, the external data is 16 bits; the internal data bus is 32 bits; the instruction and data caches are 2k bytes (16 bytes per line).

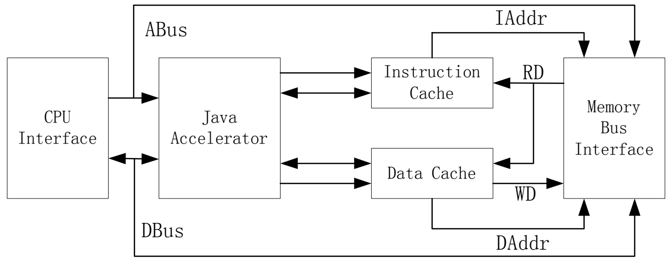


Fig. 4. Chip Architecture

The CPU interface works like an asynchronous SRAM or flash memory, and the memory bus interface only provides the asynchronous read/write control to SRAM and flash memory. In this way, except the test clock input in the scan chain for JTAG test, no other clock is needed.

The java accelerator core is basically a single issue RISC like processor. It can concurrently load 2 words from register, execute some operation, and then store one word result into register. All the operations of java bytecodes are based on java stack. We make the register file as java stack top window, so for the bytecodes based on stack top, our java accelerator can work like a RISC processor.

Bytecode instructions are variable length instructions. For hardware implementation, the instruction length ranges from 1 to 5 bytes. Direct execution of bytecodes on stack based embedded processors is invariably constrained by the limitation of the stack architecture for accessing operands. Folding is an optimization implemented in such architectures to coalesce multiple stack based instructions to a single RISC-style instruction with optimized data accessing. Thus, both power and execute time are saved.

The datapath pipeline has 6 stages:

1) F: Instruction Fetch.
2) D: Instruction Decode, check the possibility of instruction folding.
3) R: Register access, detect the exact operand address, and produce the new address for variable loading, as well as the target address.
4) E: Execution unit, arithmetic and logic operation, send read request to cache.
5) C: Cache access, receive loaded data, or send result to cache write buffer.
6) W: Write the result back into register or variable area.

For a single issue RISC like processor, bytecode instructions will be executed serially in the order as fetched. The main execution unit in datapath includes 32 bits ALU and barrel shifter.

A. *Instruction fetch unit*

Like a normal RISC pipeline, the first stage of the java accelerator is "instruction fetch". As shown before, some instructions can be folded and executed as one instruction. To detect the folding instructions, 7 bytes is adequate, including index bytes, and the biggest length should be 8 bytes. So the instruction buffer should be 16 bytes. In case that the higher 8 bytes are consumed by the folding unit, the rest half buffer will transfer to replace the bytes that have been consumed. When branch instruction encountered and new PC out of the instruction buffer, the buffer will be flushed and the new bytecode stream will be directly assigned to I[15:8] within 2 fetch operations.

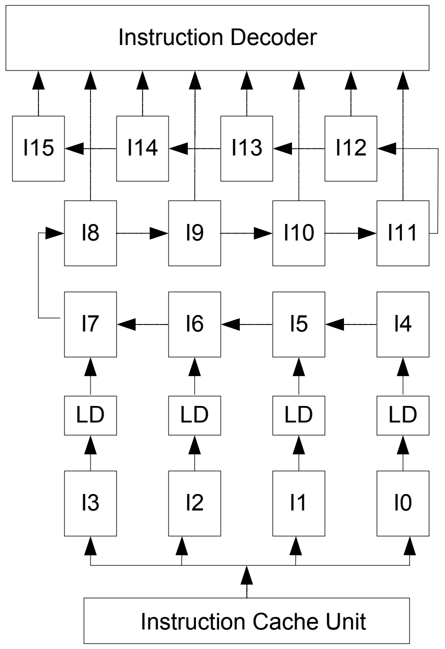


Fig. 5. Instruction Buffer Unit

In Fig. 5, LD means length decoder, which detects the length of every bytes sent from instruction fetch unit. To facilitate instruction folding in the next stage, all hardware implemented bytecode instructions are divided into 6 classes and with 6 bits to identify:

--NF (unable to be fold)
--LV (load variable)
--OP (consume top 2 stack words and push result back)
--MEM (store stack top into variable area)
--OP1 (consume 1 stack word)
--OP2 (consume 2 stack words)

When trap instruction to be executed, the java accelerator won't send interrupt signal until all other 5 pipelines are idle. When branch instruction encountered, the instruction buffer chain will halt till new PC is set.

In the instruction buffer, originally 8 bits unit will really occupy 19 bits:

1) 8 bits for bytecode instruction code
2) 6 bits to identify hardware bytecode instruction: NF, LV, OP, MEM, OP1, OP2
3) 2 bits to present PC uncertain instruction: Trap, Branch
4) 3 bits for instruction length

B. *Instruction decode unit*

The instruction decode unit gets the operation control from the higher 8 bits of the instruction buffer. For its RISC like architecture, up to 4 bytecode instructions can be executed as 1 instruction in the java accelerator. After the decode unit, 4 part signals will be sent to the next stage. (RS1, RS2: address index of 2 operands, OP_CODE: operation control signal, RD: address of destination.)

The microcode ROM is designed for the implementation of complicate bytecode as “return” instructions. In OP_CODE, one bit will show if the microcode should follow the decoded bytecodes.

Two special caches are integrated into the decode unit, where the extended branch target buffer (EBTB) cache is for branch prediction, and the decoded bytecode cache (DBC) is to improve performance and save power.

1) Instruction folding unit

There are 5 instruction folding sets:

--A constant load or a local load followed by an ALU instruction.

--An ALU operation followed by a local store.

--A constant load or local load followed by an ALU instruction followed by a local store.

--Two constant and/or local loads followed by an ALU instruction.

--Two constant and/or local loads followed by an ALU instruction followed by a local store.

When “NF” flag of the highest instruction byte is set, the folding unit will be bypassed to save power.

2) Decoded OP_CODE Format

Instruction decoder will generate direct control signals for execution units. OP_CODE is composed of 26 bits control signal:

--Push, Store, Pop: Stack operation

--Double, Quadruple: Operand or index bytes number

--Add, Sub, Mul: Arithmetic operation

--LeftShift, RightShift, Unsigned: Shift operation

--And, OR, XOR: Bitwise operation

--Convert: “i2l, l2i, i2b, i2c, i2s” are located in RS2_byte1

--ifeq, ifne, iflt, ifge, ifgt, ifle: Conditional branch operation

--SetPC: Update PC with ALU result

--SetOPTOP[3:0]: Update stack top in different mode

--Microcode: successor operation control signal will come from microcode ROM

3) RS1/RS2 and RD format

The format of RS1/RS2 and RD are shown in Fig. 6. In RS1 (The constant operand is in the format of 8 bits immediate operand):

--V1:

a) 1: RS1_Byte1:RS1_Byte2 is the offset of the variable to be loaded.
b) 0: RS1 is not variable.

--W1:

a) 1: RS1_Byte1:RS1_Byte2 is 16 bits immediate operand; if negative, sign extend to 32 bits.
b) 0: RS1 is not 16 bits operand.

--B1:

a) 1: Ignore RS1_Byte1; RS1_Byte2 is 8 bits immediate operand; if negative, sign extend to 32 bits.
b) 0: RS1 is not 8 bits operand.

In RS2, the V2, W2 and B2 bits are with the same definition.

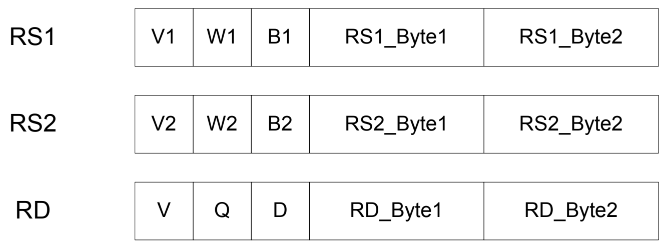


Fig. 6. Source operand and destination address after decode

In RD:

--V:

a) 1: Store operation: RD_Byte1:RD_Byte2 is the target variable offset.
b) 0: no store operation.

--Q:

a) 1: Branch operation: address is determined by 4 index bytes as offset.
b) 0: no branch with 4 bytes offset.

--D:

a) 1: Branch operation: address is determined by 2 index bytes as offset.
b) 0: no branch with 2 bytes offset.

4) Control signal of multi destination instruction

Generally, only one word will be stored in one instruction (only one write port in stack cache). But some bytecodes will store multi words into stack. Such multi destination instructions are mainly about long and double operation, duplicate, return instruction:

--Load/store const Long/Double

--Load/Store Var Long/Double

--Dup, Dup2

--Swap

--Add/Sub Long

--negate Long

--shift Long

--bitwise Long

--return operation

The multi destination instruction is controlled by microcode after decoded.

5) Decoded bytecode cache

In superscalar and VLIW processors, an intermediate organization called fill unit [4] is applied as hardware assist to compact micro-operations that are generated from sequentially fetched instructions into a decoded instruction cache. In our java accelerator, the bytecode instruction fetch, bytecode decoding and folding are definitely in the critical path, and because of the uncertainty with bytecode instruction length and consumed bytes by folding logic, the next instruction address to be fetched must wait for the computation of decode and folding logic. Both delay and power consumption is significant in this period. When bytecode loop encountered, the repeated instruction fetching, decoding and folding operation can be avoided if the execution control signal has been stored and will be read when needed.

To save time and power in loop, decoded bytecode cache (DBC) was applied to store the decoded operation control signal. Its architecture is shown in Fig. 7.

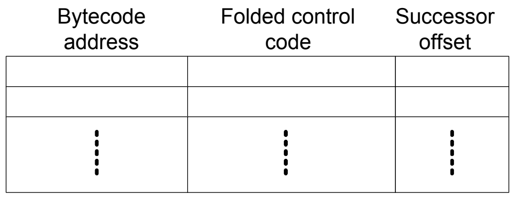


Fig. 7. Line format in DBC

In every line of DBC (above frame), the first field is "bytecode address" which is the beginning address of the decoded bytecode instruction group, followed by the exact control signal next pipeline stage needed, and the last is the address offset of successor bytecode instruction or the consumed bytecode bytes in decoder.

DBC only stores the bytecodes in program loop. When the decoder gets the consumed bytes and generates the new bytecode address, DBC will be sent with this address. If hit, "folded control code" will be read out and sent to the next pipeline stage. The successor offset can be used to generate the new fetch address and certainly such address should be checked in DBC first.

For the control codes fetched from DBC instead of the long way from memory to decoder, more time can be used to search the match within DBC. So a large DBC is tolerant. The miss penalty of DBC is the delay of DBC itself. We invoke DBC when branch backward happened, and update DBC when backward branch taken for a second time. When in nested loops, if the internal loop was already in DBC, it's not necessary to write it into DBC again. (The begin and end address of every loop is also appended).

The DBC is organized as a 64 entry 2-way associated cache.

6) Branch prediction

When conditional branch instruction encountered, the processor pipeline will be stalled till the branch condition has been reached. So the branch prediction accuracy is a major performance-affecting factor in pipelined processor. To decrease the additional delay, both static and dynamic branch prediction have been applied in modern processor design. The static branch prediction is quite simple with the strategy that branch taken when backward and not taken when forward. The dynamic branch prediction needs specified hardware logic to store the branch history.

According to the instruction statistic from benchmark in Tab. 1, the conditional branch instruction is undoubtedly a significant part in java application [5]. To improve the performance in our java processor and match the cost constraint, we propose a branch prediction scheme to work with DBC, which is suitable for bytecode instruction set.

Tab. 1. Statistic of branch instruction in SPEC JVM98

	Branch	**Total Bytecodes**
compress	6.1%	951990234
Jess	9.6%	8126332
Db	10.2%	2035798
Javac	8.6%	5958654
Mpegaudio	8.4%	115748387
Mtrt	5.1%	50683565
jack	11.0%	175740325

Dynamic branch prediction uses information gathered during the run-time of the program to predict branch direction. The techniques, such as branch target buffers (BTB), pattern history table (PHT), branch target address cache (BTAC), to keep track of the direction branch is likely to take. The implementation of dynamic branch prediction requires dedicated hardware and sizeable chip area, thus its cost is big. However, the more expensive, the better performance of the processor [6].

BTAC is a set of associative memory, in which each line contains: the address of branch or jump instruction, the most recent target address for that branch or jump, the information that permit a prediction as to whether or not the branch will be taken. To keep the size of BTAC small, only predicted taken branch addresses are stored. BTB is an extension to BTAC, not only the branch target address is stored but also the target instruction itself.

For variable length bytecodes, we modify BTB and make the new prediction unit to match the character of bytecodes. We call it Extended BTB (EBTB). As shown in Fig. 8, each line of EBTB contains the branch address (after instruction folding), folded control code (as in DBC), branch offset, successor offset (if no branch taken), and prediction bits. The EBTB will be organized as a 64 entry full associated cache. For embedded applications, this EBTB can keep track of the most recent branch operations.

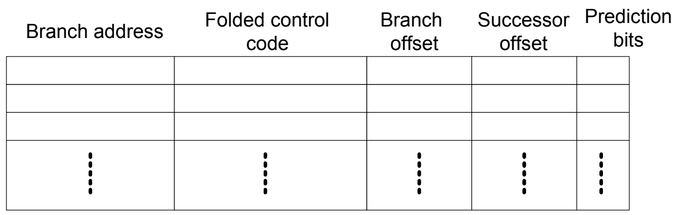


Fig. 8. Line format of EBTB

EBTB runs with DBC. The branch instruction stored in EBTB is the bytecodes that has been folded. The folded branch instruction will only be stored in EBTB, so there is no conflict between the branch addresses of EBTB and DBC. The combination of the two caches is shown in Fig. 9.

After consumed bytes length computed by the decoder, the new PC will be compared against the address in EBTB and DBC. If hit, the folded control code in EBTB or DBC will be read and sent to the next pipeline stage when it's ready. Simultaneously, branch target address from EBTB or next address from DBC will be read and sent back to compare if the new address is still in the two caches. Only when the new PC is not in EBTB and DBC, it will be sent to fetch unit. For bytecode stream loops, such scheme will eliminate the need of memory access to the least.

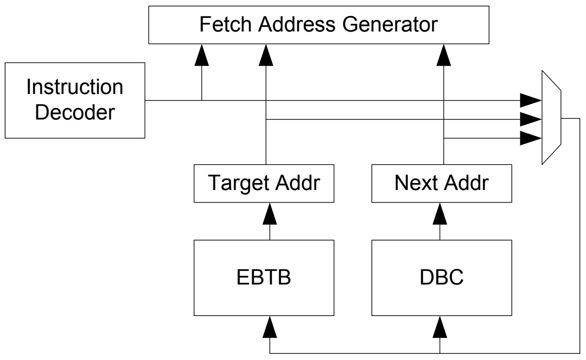


Fig. 9. Address generator for instruction fetch

There are two bits in prediction bits field, so 4 states map the direction of branch: strongly taken, weakly taken, weakly not taken, strongly not taken. As shown in Fig. 10, the state machine is the same as UltraSPARC scheme.

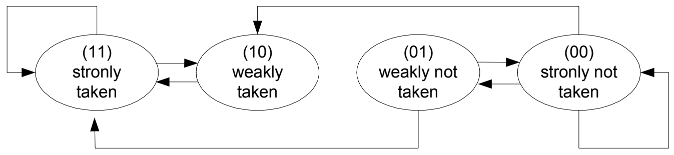


Fig. 10. State machine of prediction bits

As shown in Fig. 11, the two-bit predictor scheme uses only the recent behavior of a single branch to predict the future of that branch. The exact accuracy statistic is underway, but the advantage of this prediction scheme is evident.

C. *Register access unit*

Although most operands are from the stack cache register, there are still some from local variable area outside the stack cache. When loading variables, the real location should be checked. If it's not in stack cache, it needs to be read from cache or main memory.

The stack cache register module has 2 read ports and 1 write port. It can concurrently read 2 operands and simultaneously write 1 word into register. That is the reason to fold multi-bytecode into one instruction.

Invoking methods in java is expensive as it requires the setting up of an execution environment and a new stack for each new method.

1) *Java accelerator control registers*

--Program Counter Register (PC)
--Constant Pool Base Pointer Register (CONST_POOL)
--Java accelerator control register
--ST_Limit: the lowest address for java stack
--Processor Status Register (PSR): half bits are for trap handler address
--Data segment address offset register
--Instruction segment offset register
--Data mask register
--Instruction mask register
--Thread frame register
--Thread S_VAR register
--SC_TOP register
--SC_Bottom register
--Breakpoint register

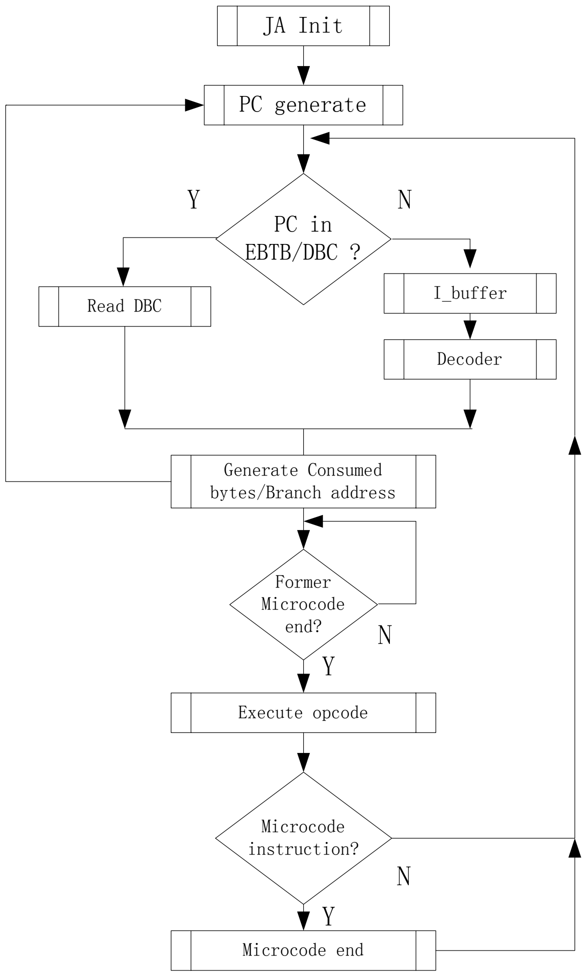


Fig. 11. Workflow of PC generation

2) *Operand dependency*

Operand dependency needs to be detected before loading word from stack cache. If the operand locates in the address that the former instruction writes back, the operand will be loaded from ALU result temporary register, not from stack cache. There are 3 line FIFO queues followed ALU to store temporary ALU results.

When load from memory operation encountered, a "NOP" instruction will be inserted to wait until the required data has been stored into the temporary register.

3) *Detection of register operand access*

In 64 stack cache registers, address point to stack top is just the 6 LSB (least significant bits) of 32 bits SC_TOP. The 6 bits stack cache bottom address mapping to SC_Bottom is also 6 LSB of SC_Bottom.

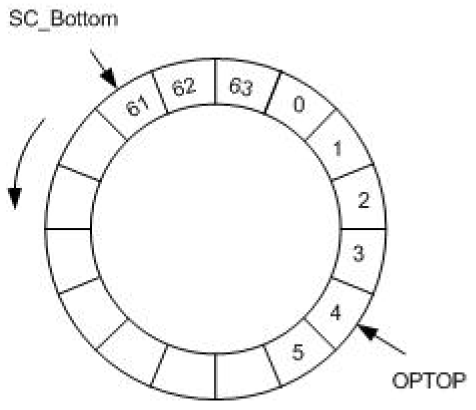


Fig. 12. Stack Cache Register

As shown in Fig. 12, with the method invoke or return, the stack cache will overflow or underflow. If overflow or underflow is detected, the pipeline will halt, and then move register contents into cache memory when overflow, or add more stack contents flow SC_Bottom when underflow. Such transfer between stack cache register and cache memory should be executed automatically, and every time up to 4 words should be transferred because 4 words compose 1 line.

D. Cache access interface

In the first version our java accelerator, the direct mapped cache is integrated. Since the core SRAM can be generated by memory compile tools, it's easy to implement it in layout and extract the timing parameters. The basic frame is shown in Fig. 13.

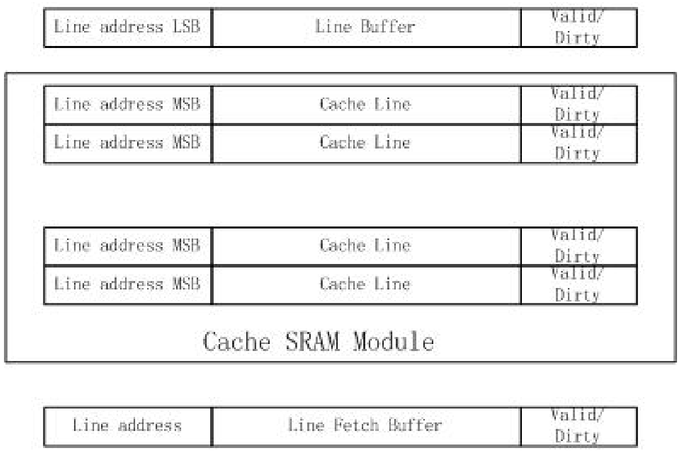


Fig. 13. Direct mapped cache module

When fetching a line from main memory, the new line will be stored in "Line Fetch Buffer", and then send the required word into the java accelerator. Because of no access to SRAM module, no precharge operation is needed. Thus, power is saved.

When fetching word from SRAM module, the corresponding line will be fetched into "Line Buffer". If the followed read and write are only focused on this line, no precharge is needed too. So power is saved.

To keep cache coherence, the data cache in the java accelerator should be designed in "write through" strategy, which means when the accelerator writes data into cache; it will also write the data into memory.

E. Pipeline Control

The pipeline of our java accelerator is shown in Fig.14. The pipeline halts when the following conditions encountered:

1) Trap bytecode instruction

If trap instruction is detected in "instruction fetch", the "IF" stage will halt. After all successor pipeline stages idle, the java processor will send INT signal to the main processor.

2) Stack cache underflow/overflow

The stack cache is the slide head window for stack in memory. If more operands than permitted need to be pushed into stack, it will move the old register content to corresponding memory unit. Or if too few registers hold the stack content, it will load more sequential stack from memory. When the stack cache exchanges data between the java accelerator and memory, stages following "RC" will halt.

3) Breakpoint

When PC value matches the breakpoint, which is set in initial state, the later stages of RC will continue till pipeline halts.

4) External halt

When external hold signal is set, all pipeline stages will halt until the external hold signal cleared.

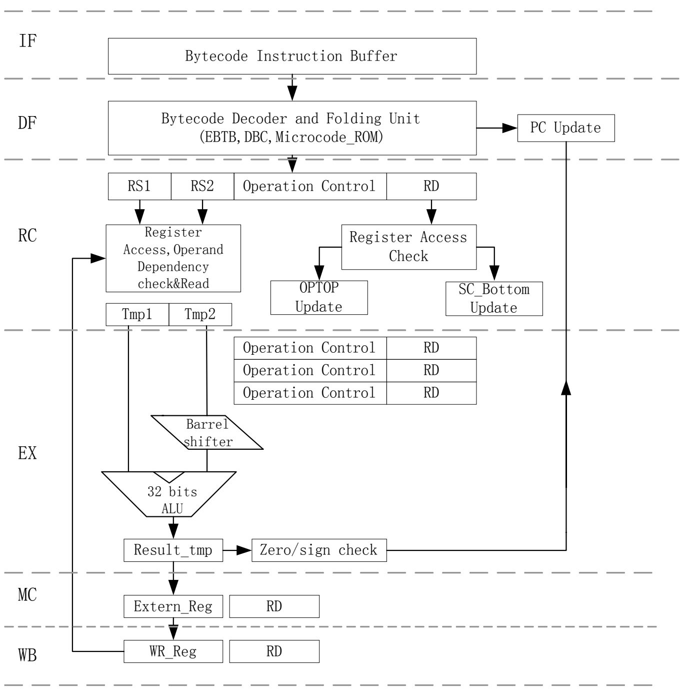


Fig. 14. Pipeline and datapath

V. System Simulation

Before the final java accelerator chip manufactured, the java application was running on a virtual environment. The virtual platform was constructed based on ARMulator, the emulator of ARM processor in instruction level. The java accelerator was constructed with SystemC language. The RTOS was based on Uclinux, and the JVM was modified from Kaffe [7].

The optimizing target is critical low power consumption. We set different weight values for separate operations. Because the java accelerator is designed in asynchronous circuits, the power consumption resides in functional components. For the whole system, the power consumption can be estimated. The power consumption difference between JIT compiler JVM and our hardware accelerated JVM has been researched. In our case, significant power consumption resides in memory read/write operations. The SystemC accelerator emulator works as a child process invoked by JVM.

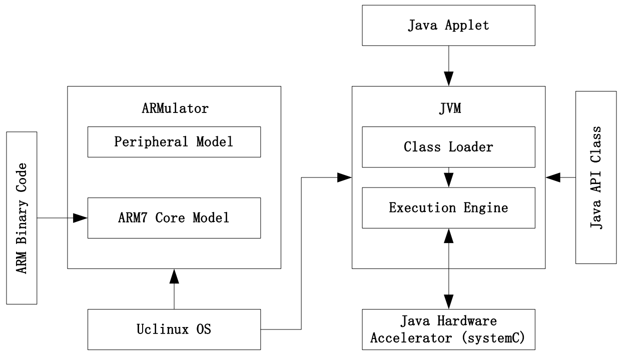


Fig. 15. Virtual JVM Platform

In the virtual environment in Fig. 15, HW/SW co-design was carried on. To monitor RTOS running is difficult in a real hardware platform, but in a virtual software platform, every program step can be traced easily.

VI. Conclusion and Future Work

The scheme proposed in this paper is suitable for accelerating java execution for pervasive computing. The simulation results of small applets show that the combining of EBTB and DBC provides a new solution for java processor design.

To further minimize the power consumption, more research should be done to restrain the concurrent work of EBTB and DBC, and FP unit should be integrated to minimize the control transfer between the main processor and java accelerator.

References

[1] Ramesh Radhakrishnan, etc, "Java Runtime Systems: Characterization and Architectural Implications", *IEEE Transactions on computers*, Vol. 50, No. 2, Feb. 2001.

[2] M. Watheq El-kharashi, Josh Pfrimmer, etc. , "A Design Space Analysis of Java Processors", in *Proc. IEEE Pacific Rim 2003 Conference*, Victoria, Canada, August 2003.

[3] Balsa. See http://www.cs.man.ac.uk/apt/projects/balsa/index.html.

[4] Ramesh Radhakrishnan, Deependra Talla and Lizy Kurian John, "Allowing for ILP in an Embedded Java Processor," in *Proc. the 27th International Symposium on Computer Architecture*, June 2000.

[5] The Standard Performance Evaluation Corporation. SPEC JVM98 benchmarks. See http://www.specbench.org/osg/jvm98.

[6] Karthik Thangarajan, Wagdy Mahmoud, etc, "Survey of branch prediction schemes of pipelined processors", in *Proc. the Thirty-Fourth Southeastern Symposium on System Theory*, Alabama, March 2002.

[7] Kaffe JVM. See http://www.kaffe.org.

Paper IX

Formal Verification of a Ubiquitous Hardware Component

L. Yan

Available in: *Post-Proceedings of the First International Conference on Embedded Software and System (ICESS'04), 2005. Lecture Notes in Computer Science 3605, Springer-Verlag.*

Extended version available as: *L. Yan. Formal Verification of a Ubiquitous Hardware Component. TUCS Technical Reports, No. 637, Turku Centre for Computer Science, Finland, 2004.*

Formal Verification of a Ubiquitous Hardware Component

Lu Yan
Turku Centre for Computer Science
Lemminkäisenkatu 14 A, 20520 Turku, Finland
lyan@abo.fi

TUCS Technical Report
No 637, November 2004

Abstract

The paper begins by discussing various approaches to hardware specification and verification. The main emphasis is on using mechanical verification tools to assist the verification process. The case study is the verification of a seven-segment LED display decoder circuit design, in which two popular verification tools, HOL and PVS, are compared and evaluated.

Keywords: Hardware verification, formal methods, ubiquitous systems

TUCS Laboratory
Distributed Systems Design Laboratory

1 Introduction

The development of microelectronics has allowed hardware designers to build remarkably complex devices. However, it becomes increasingly difficult to ensure these devices free of design errors. In most cases, exhaustive simulation of a medium size design is impossible and the correctness of the design cannot be assured. This is a serious problem in safety-critical applications, where a small design error may cause loss of life and extensive damage. Even in the case where safety is not the primary concern, a design error means costly and time-consuming rechecking in massive production lines.

A solution to the problem is to apply formal methods for verification of correctness of hardware designs - hardware verification. With this approach, the behavior of hardware is described mathematically, and formal proof is used to verify the intended behavior. The proofs can be very large and complex, so mechanical verification tools are often used to assist the verification.

We illustrate our experiences with formal verification in ubiquitous hardware design via a comparative case study of the verification of a circuit design of seven-segment LED display decoder: A seven-segment LED display is comprised of seven light emitting diodes (LED). Input signals are applied to the input port of the seven-segment decoder, and the decoder translates them into ON/OFF status of the seven LEDs. Then, selected combinations of the LEDs are illuminated to display numeric digits and other symbols.

2 What is formal hardware verification

We consider a formal hardware verification problem to consist of *formally establishing that an implementation satisfies a specification.* The term *implementation (Imp)* refers to the hardware design that is to be verified. This entity can correspond to a design description at any level of the hardware abstraction hierarchy, not just the final physical layout (as is traditionally regarded in some areas). The term *specification (Spec)* refers to the property with respect to which correctness is to be determined. It can be expressed in a variety of ways - as a behavioral description, as an abstracted structural description, as a timing requirement etc.

In particular, we do not address directly the problem of specification validation, i.e. whether the specification means what it is intended to mean, whether it really expresses the property one desires to verify, whether it completely characterizes correct operation etc. A specification for a particular verification problem can itself be made the object of scrutiny, by serving as an implementation for another verification problem at a conceptually higher level. Similarly, at the lowest end too, we do not specifically address the problem of model validation, i.e. whether the model used to represent the implementation is consistent, valid, correct etc. It is obvious that the quality of verification can only be as good as the quality of the models used.

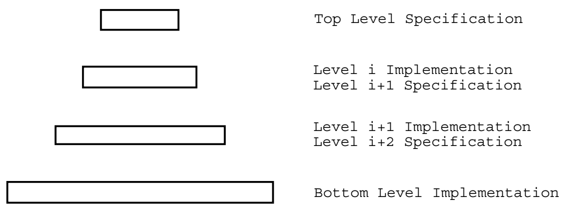


Figure 1: Hierarchical verification[1]

An important feature of the above formulation is that it admits hierarchical verification corresponding to successive levels of the hardware abstraction hierarchy. Typically, the design of a hardware system is organized at different levels of abstraction, the topmost level representing the most abstract view of the system and the bottommost being the least abstract, usually consisting of actual layouts. Verification tasks can also be organized naturally at these same levels. An implementation description for a task at any given level, serves also as a statement of the specification for a task at the next lower level, as shown in Figure 1. In this manner, top-level specifications can be successively implemented and verified at each level, thus leading to implementation of an overall verified system. Hierarchical organization not only makes this verification process natural, it also makes the task tractable. By breaking this large problem into smaller pieces that can be handled individually, the verification problem is made more manageable. It effectively increases the range of circuit sizes that can be handled in practice.

2.1 Hardware verification method

Two things are needed for any method of hardware verification based on rigorous specification and formal proof. The first is a formal language for describing the behaviors of hardware and expressing proposition about it. The ideal language is expressive enough to describe hardware in a natural concise notation yet still has a well-understood and reasonably simple semantic. The second requirement is a deductive calculus for proving propositions expressed in this language. It must be logically sound and it should be powerful enough to allow one to prove all the true propositions about hardware behavior that arise in practice.

Various formal languages and associated proof techniques have been proposed as a basis for hardware verification. These range from special-purpose hardware description languages with *ad hoc* proof rules to systems of formal logic and subsets of ordinary mathematics. Formal methods for reasoning about hardware behavior have been based, for example, on algebraic techniques, various kinds of temporal logic, functional programming techniques, predicate calculus, and higher order logic.

The details of the verification methods based on these different formalisms vary, but many of them share a common general approach. This typically involves the following four steps:

1. Write a formal specification S to describe the behavior that the device to be verified must exhibit for it to be considered correct.

2. Write a specification for each kind of primitive hardware component used in the device. These specifications are intended to describe the actual behavior of real hardware components.

3. Define an expression D which describes the behavior of the device to be proved correct. The definition of D has the general form

   $$D = P_1 + \cdots + P_n$$

   where $P_1, \cdots, Pn$ specify the behavior of the constituent parts of the device and $+$ is a composition operator which models the effect of wiring components together. The expressions $P_1, \cdots, Pn$ used here are instances of the specifications for primitive devices defined in step 2.

4. Prove that the device described by the expression D is correct with respect to the specification S. This is done by proving a theorem of the form

   $$\vdash D \quad \text{satisfies} \quad S$$

   where 'satisfies' is some satisfaction relation on specifications of hardware behavior. This correctness theorem asserts that the behavior described by D satisfies the specification of intended behavior S.

When the device to be proved correct is large, this method is usually applied hierarchically. The design is structured into a hierarchy of components and sub-components, and specifications that describe primitive components at one level of the hierarchy then become specifications of intended behavior at the next level down. The structure of the proof mirrors this hierarchy: the top-level specification is shown to be satisfied by an appropriate connection of components; at the next level down, each of these components is shown to be correctly implemented by a connection of sub-components, and so on down to the lowest level, where the components used correspond to devices available as hardware primitives.[31]

2.2 Hardware verification using higher order logic

The version of higher order logic described here was developed by Mike Gordon at the University of Cambridge. The main difference between first order logic and higher order logic is that higher order logic allows quantification over predicates. The ability to quantify over predicate symbols leads to a greater power of expressiveness in higher order logic. Another significant difference is that higher order logic admits higher order predicates and functions, i.e. arguments and results of predicates and functions can themselves be predicates or functions. This allows functions to be manipulated just like ordinary values, which leads to a more mathematically elegant formalism.

The following short description of higher order logic is not complete, although it covers important notations of the logic, which provides some background information for the later example. A full description of higher order logic can be found at [17].

Types Higher order logic is a typed logic. The syntax of types in higher order logic is given by

$$\sigma ::= c|v|(\sigma_1, \ldots, \sigma_n)op$$

where $\sigma, \sigma_1, \ldots, \sigma_n$ range over types, c ranges over type constants, v ranges over type variables, and op ranges over n-ary type operators.

Terms The notation of terms in higher order logic can be viewed informally as an extension of the conventional syntax of predicate calculus in which variables can range over functions and functions can take functions as arguments or yield functions as results. The syntax of terms in higher order logic is given by

$$M ::= c|v|(MN)|\lambda v.M$$

where c ranges over constants, v ranges over variables, and M and N range over terms.

Sequents, theorems and inference rules A sequent is written $\Gamma \vdash P$, where Γ is a set of boolean terms called assumptions and P is a boolean term called the conclusion. When the set Γ is empty, the notation $\vdash P$ is used. In this case, P is a formal theorem of the logic. The same notation is used for the axioms of the logic. All inference rules of the logic can be found at Table 1.

The approach to specifying hardware behavior in higher order logic is to specify the behavior of a device by describing the combinations of values that can be observed on its external wires. A specification is expressed formally in logic by a boolean-valued term whose free variables correspond to these external wires. This term imposes a constraint on the values of these variables. To reflect the behavior of the device it specifies, the term is chosen so that the combinations of values that satisfy this constraint are precisely those which can be observed simultaneously on the corresponding external wires of the device itself.

As an example, consider the device **Dev** shown below.

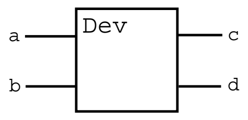


Table 1: Inference rules of higher order logic

1.	ASSUME: $\frac{}{\{P\}\vdash P}$
2.	REFL: $\frac{}{\vdash N=N}$
3.	BETA_CONV: $\frac{}{\vdash(\lambda v.N)M=N[M/v]}$
4.	ABS: $\frac{\Gamma\vdash M=N}{\Gamma\vdash(\lambda v.M)=(\lambda v.N)}$ (v not free in Γ)
5.	INST_TYPE: $\frac{\Gamma\vdash P}{\Gamma\vdash P[\sigma_1,\ldots,\sigma_n/\alpha_1,\ldots,\alpha_n]}$
6.	DISCH: $\frac{\Gamma\vdash P}{\Gamma-\{Q\}\vdash Q\supset P}$
7.	MP: $\frac{\Gamma_1\vdash P\supset Q \quad \Gamma_2\vdash P}{\Gamma_1\cup\Gamma_2\vdash Q}$
8.	SUBST: $\frac{\Gamma_1\vdash N_1' \quad \ldots \quad \Gamma_n\vdash N_n=N_n' \quad \Gamma\vdash P}{\Gamma_1\cup\ldots\cup\Gamma_n\cup\Gamma\vdash P[N_1',\ldots,N_n'/N_1,\ldots,N_n]}$

This device has four external wires: a, b, c, and d. A specification of its behavior in logic is therefore a boolean-valued term of the form $S[a, b, c, d]$, constructed so that for all values of the free variables a, b, c and d:

$$S[a,b,c,d] = \begin{cases} \mathbf{T} & \text{if the values } a, b, c, \text{ and } d \text{ could occur} \\ & \text{simultaneously on the corresponding} \\ & \text{external wires of the device } \mathbf{Dev} \\ & \\ \mathbf{F} & \text{otherwise} \end{cases}$$

This approach to specifying hardware describes its behavior only in terms of the values that can be observed externally. No information about internal state is used in a specification. Furthermore, there is no distinction between the inputs and the outputs of a device; the constraint imposed by a specification on its free variables need not be a functional one. Of course, the free variables in a specification need not stand for the values on the physical wires of an actual circuit; they may represent more abstract externally observable quantities. Both specifications of hardware primitives and specifications of the intended behavior of designs can therefore be expressed by this method.

Once a design has been constructed, its correctness can be expressed by a proposition which asserts that this design in some sense satisfies an appropriate specification of required or intended behavior. The most direct way of formulating this satisfaction of a design is asserted by a theorem of the form

$$\vdash D[v_1, \ldots, v_n] = S[v_1, \ldots, v_n],$$

Where the term $D[v_1, \ldots, v_n]$ is the design of the device asserted to be correct and the term $S[v_1, \ldots, v_n]$ is the specification of required behavior. This theorem states that the truth-values denoted by these two terms are the same for any assignment of values to the free variables $v_1, \ldots, v_n$. This is usually appropriate for small and relatively simple hardware designs; for more complex designs, it is often impractical to express correctness this way, because in most real products, any

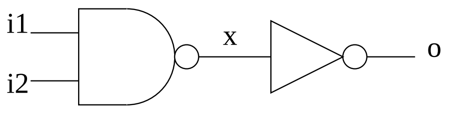


Figure 2: Implementation of two input AND gate

complete logically equivalent specification is likely to be too large and complex to reflect the designer's intent. Hence it is wise to build a partial specification for the design. In this case, the satisfaction relation used to express correctness must therefore express a relationship between a strong constraint (design) and weaker one (specification) rather than strict equivalence. Suppose that $D[v_1, \ldots, v_n]$ and $S[v_1, \ldots, v_n]$ are the design of the device and the partial specification of required behavior respectively. We can formulate this satisfaction relationship as

$$\vdash D[v_1, \ldots, v_n] \Rightarrow S[v_1, \ldots, v_n].$$

2.3 A small example

The basic idea of this approach is to embed both implementation and the specification in higher order logic. The correctness statements, like that every behavior of the implementation satisfies the specification, are then cast in terms of proving some relation in higher order logic. To illustrate this process, we use a trivial example to show many of the underlying ideas.

The task is to show that assuming the NAND gate and the NOT gate behave as specified, then combining them as in Figure 2 yields a two input AND gate. In order to achieve this, we need to carry out four steps:

1. Specify the implementation of the AND gate.
2. Specify the behavioral models for the NAND and NOT gates.
3. Specify the intended behavior of the AND gate.
4. Prove that the implementation satisfies its intended behavior.

There are several ways to specify the implementation of the AND gate in higher order logic. The most common way of doing this is using existential quantification to 'hide' the internal connections, and we would get:

$$\vdash AND\_IMP(i1, i2, o) = \exists x. NAND(i1, i2, x) \wedge NOT(x, o).$$

The behavioral model of the NAND and NOT gates can also be done in many ways in higher order logic. Furthermore, different behavioral models can be used depending on the amounts of details needed or desired. Here, we will use a simple zero-delay model of their behavior. Hence,

Table 2: Proof in higher order logic

Step	Proof	Explanation
0	$AND\_IMP(i1, i2, o)$	assumption
1	$\exists x.NAND(i1, i2, x) \wedge NOT(x, o)$	by def. of AND_IMP
2	$NAND(i1, i2, x) \wedge NOT(x, o)$	strip off $\exists x.$
3	$NAND(i1, i2, x)$	left conjunct of 2
4	$x = \neg(i1 \wedge i2)$	by def. of $NAND$
5	$NOT(x, o)$	right conjunct of 2
6	$o = \neg x$	by def. of NOT
7	$o = \neg(\neg(i1 \wedge i2))$	substitution 4 into 6
8	$o = (i1 \wedge i2)$	simplify using $\neg(\neg(t)) = t$
9	$AND\_SPEC(i1, i2, o)$	by def. of AND_SPEC

$$\vdash NAND(i1, i2, o) = o = \neg(i1 \wedge i2),$$
$$\vdash NOT(i, o) = o = \neg i.$$

In a similar way, the desired behavior of the AND gate can be written as

$$\vdash AND\_SPEC(i1, i2, o) = o = i1 \wedge i2.$$

We are now faced with the task of formally proving that the implementation satisfies the specification. Before we do this, however, we need to define what it means for an implementation to satisfy some specification. Again, there are several ways of expressing this. In this case, we choose to verify that the behavior of the implementation implies the behavior of the specification; thus we want to verify

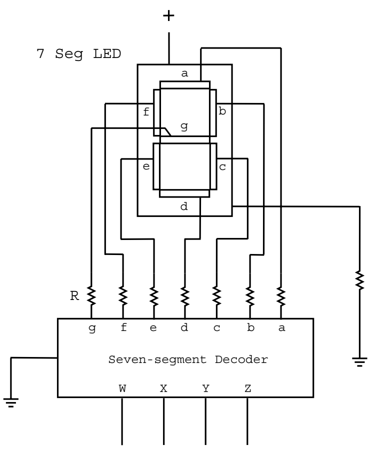
$$AND\_IMP(i1, i2, o) \Rightarrow AND\_SPEC(i1, i2, o)$$

is a valid theorem. A ’hand proof’ of this result might look like Table 2.

Although the above manual proof may appear tedious, it is still much shorter than the complete formal proof. The above example is also very simple. However, since we have the full expressive power of higher order logic at our disposal, it is quite simple to generalize the behavioral model for the individual components. In this way, delays and delay models, for example, can be introduced. Of course, the more complex the behavior model is, the more complex the correctness proof will be.[3]

3 The ubiquitous hardware

We illustrate our approach by a case study of the verification of a circuit design of seven-segment LED display decoder [20] [30] as shown below.



A seven-segment LED display is comprised of seven light emitting diodes (LED). Input signals are applied to the input port of the seven-segment decoder, and the decoder translates them into ON/OFF status of the seven LEDs. Then, selected combinations of the LEDs are illuminated to display numeric digits and other symbols.

3.1 From description to specification

The primary function of the decoder is to turn on/off corresponding LEDs based on inputs. Let W, X, Y, Z represent the input port of the decoder, then we get sixteen possible combinations of the four input signals, which means any digit (0 - 9) and some letters (A - F) can be displayed on the seven-segment LED display. Let a, b, c, d, e, f, g represent the output port of the decoder, and let on be 1 and off be 0, then we can create a truth table like Table 3 for describing the intended behavior of the decoder.

3.2 From specification to implementation

Intuitively, the abstraction of seven-segment decoder is a four-input seven-output switching function. One possible approach is to build up the circuit directly from the specification, but here we consider another approach based on partition-and-merge algorithm.[28][5]

First we divide the four-input seven-output switching function into seven four-input one-output normal functions, then implement each function separately. When all functions are ready, we put together all parts and get the final implementation. In this way, the complexity of the design task is greatly reduced. The drawback is probably some redundancy, but this can be refined in the final merging stage.

We shall go into more details of the implementation of one part as an example. The representation function is the abstraction of the intended behavior of LED a, which takes four input signals W, X, Y, Z and generate one output signal a correspondingly.

Table 3: Truth table for the switching function

Display	Input ($W\,X\,Y\,Z$)	Output ($a\,b\,c\,d\,e\,f\,g$)
0	0000	1111110
1	0001	0110000
2	0010	1101101
3	0011	1111001
4	0100	0110011
5	0101	1011011
6	0110	1011111
7	0111	1110000
8	1000	1111111
9	1001	1111011
A	1010	1110111
B	1011	0011111
C	1100	1001110
D	1101	0111101
E	1110	1001111
F	1111	1000111

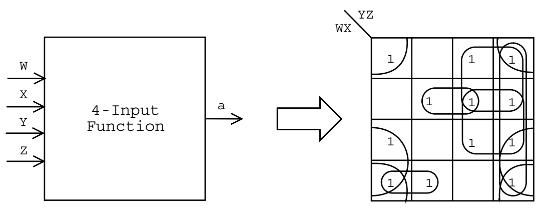


Figure 3: Karnaugh map for 4-input function

1. The first step is to build up the truth table like Table 4 for this four-input one-output function. This is actually a reduced version of Table 3.

2. With the truth table, we can get a logic expression directly.

   $a = \overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z} + \overline{W}\,\overline{X}\,Y\,\overline{Z} + \overline{W}\,\overline{X}\,Y\,Z + \overline{W}\,X\,\overline{Y}\,Z + \overline{W}\,X\,Y\,\overline{Z} + \overline{W}\,X\,Y\,Z + W\,\overline{X}\,\overline{Y}\,\overline{Z} + W\,\overline{X}\,\overline{Y}\,Z + W\,\overline{X}\,Y\,\overline{Z} + W\,X\,\overline{Y}\,\overline{Z} + W\,X\,Y\,\overline{Z} + W\,X\,Y\,Z$

3. With the initial implementation, we can refine it with emphasis on reducing the sum of minimal terms[13] in order to minimize hardware resource usage. The most common approach is to use a Karnaugh map to achieve this kind of refinement.[2][14] The process is illustrated in Figure 3.

4. After refinement, we get a more concise logic expression:

$$a = Y\,\overline{Z} + \overline{X}\,\overline{Z} + \overline{W}\,Y + X\,Y + W\,\overline{Z} + \overline{W}\,X\,Z + W\,\overline{X}\,\overline{Y}$$

Table 4: Truth table of 4-input function

Input ($WXYZ$)	Output (a)
0000	1
0001	0
0010	1
0011	1
0100	0
0101	1
0110	1
0111	1
1000	1
1001	1
1010	1
1011	0
1100	1
1101	0
1110	1
1111	1

5. Although this refinement result is good enough, we should also consider more practical issues like technology, cost, etc.. Here we choose to make the design mainly with NAND gates.

$$a = \overline{\overline{Y\,\overline{Z}} \cdot \overline{\overline{X}\,\overline{Z}} \cdot \overline{\overline{W}\,Y} \cdot \overline{X\,Y} \cdot \overline{W\,\overline{Z}} \cdot \overline{\overline{W}\,X\,Z} \cdot \overline{W\,\overline{X}\,\overline{Y}}}$$

6. Now it is time to translate the refinement result into schematic design. The diagram is straight forward, as shown in Figure 4.

7. The final step is to design the real circuit based on the schematic design. Here we choose the NAND gate model and the tool discussed in [27][33]

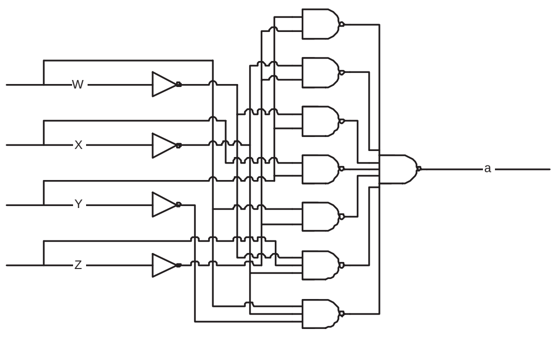


Figure 4: Schematic design for 4-input function

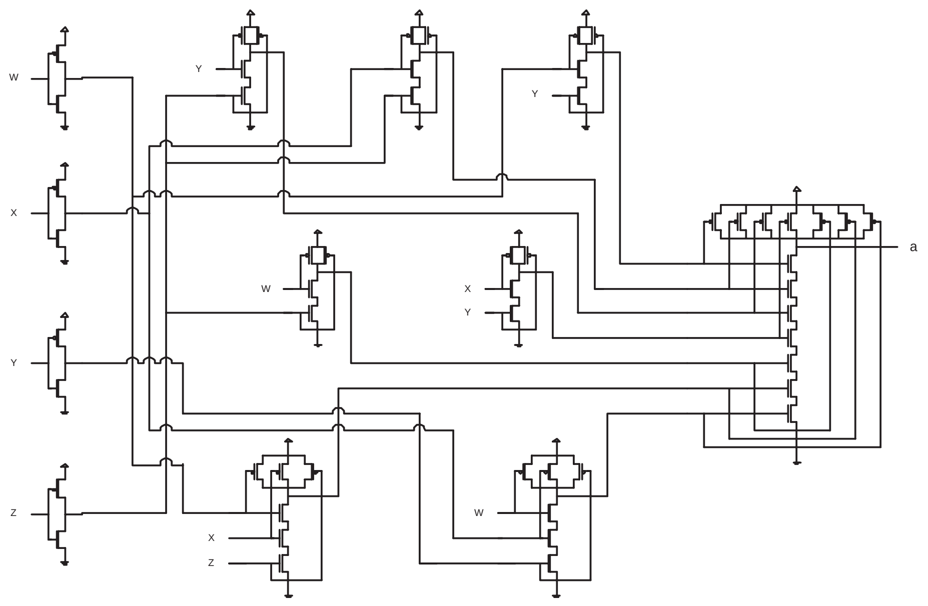


Figure 5: Circuit design for 4-input function

as the atomic element to build up the whole design. The result is shown in Figure 5.

Now the design task of the first part is completed. With the same method, we can design the other six parts:

$$b = \overline{\overline{\overline{W}\,\overline{X}} \cdot \overline{\overline{X}\,\overline{Z}} \cdot \overline{\overline{W}\,\overline{Y}\,\overline{Z}} \cdot \overline{\overline{W}\,Y\,Z} \cdot \overline{W\,\overline{Y}\,Z}}$$

$$c = \overline{\overline{W\,\overline{X}} \cdot \overline{\overline{Y}\,Z} \cdot \overline{\overline{W}\,X} \cdot \overline{\overline{W}\,\overline{Y}} \cdot \overline{\overline{W}\,Z}}$$

$$d = \overline{\overline{W\,\overline{Y}} \cdot \overline{\overline{W}\,\overline{X}\,\overline{Z}} \cdot \overline{X\,\overline{Y}\,Z} \cdot \overline{X\,Y\,\overline{Z}} \cdot \overline{\overline{X}\,Y\,Z}}$$

$$e = \overline{\overline{W\,X} \cdot \overline{Y\,\overline{Z}} \cdot \overline{W\,Y} \cdot \overline{\overline{X}\,\overline{Z}}}$$

$$f = \overline{\overline{\overline{Y}\,\overline{Z}} \cdot \overline{W\,\overline{X}} \cdot \overline{W\,Y} \cdot \overline{X\,\overline{Z}} \cdot \overline{\overline{W}\,X\,\overline{Y}}}$$

$$g = \overline{\overline{W\,\overline{X}} \cdot \overline{Y\,\overline{Z}} \cdot \overline{W\,Z} \cdot \overline{\overline{X}\,Y} \cdot \overline{\overline{W}\,X\,\overline{Y}}}$$

We notice that remaining parts are very similar to the first part in logic expressions, which will also lead to very similar system infrastructures. In order to keep the text concise, we don't list down the designs of the other six parts due to the similarity in these design results.

4 Verification in HOL

HOL is a general theorem proving system developed at the University of Cambridge that is based on higher order logic. HOL is not a fully automated theorem prover but is more than simply a proof checker, falling somewhere between these two extremes. HOL has several nice features as a verification environment:

- Several built-in theories, including booleans, individuals, numbers, products, sums, lists, and trees. These theories build on the five axioms that form the basis of higher order logic to derive a large number of theorems that follow from them.
- Rules of inference for higher order logic. These rules contain not only the eight basic rules of inference from higher order logic, but also a large body of derived inference rules that allow proofs to proceed using larger steps. The HOL system has rules that implement the standard introduction and elimination rules for Predicate Calculus as well as specialized rules for rewriting terms.
- A large collection of tactics. Examples of tactics include `REWRITE_TAC` which rewrites a goal according to some previously proven theorem or definition, `GEN_TAC` which removes unnecessary universally quantified variables from the front of terms, and `EQ_TAC` which says that to show two things are equivalent, we should show that they imply each other.
- A proof management system that keeps track of the state of an interactive proof session.
- A metalanguage, ML, for programming and extending the theorem prover. Using the metalanguage, tactics can be put together to form more powerful tactics, new tactics can be written, and theorems can be aggregated to form new theories for later use. The metalanguage makes the verification system extremely flexible.

4.1 HOL Overview

The logic of the HOL system is built on higher order logic. The core of the system is rather small. It is built on 5 axioms (Table 5) and 8 rules of inference (Table 6).

The HOL theorem prover uses an ASCII approximation (Table 7) to standard logic notation. One of the types that have been declared is the type of terms in the HOL logic. To enter the term which is a conjunction of two boolean variables A and B, just type the following:

```
- Term `A /\ B`;
> val it = ``A /\ B`` : term
```

Table 5: HOL axioms

1.	BOOL_CASES_AX
	$\vdash \exists b : bool.(b = \mathrm{T}) \vee (b = \mathrm{F})$
2.	IMP_ANTISYM_AX
	$\vdash \exists b_1 b_2.(b_1 \Rightarrow b_2) \Rightarrow (b_2 \Rightarrow b_1) \Rightarrow (b_1 = b_2)$
3.	ETA_AX
	$\vdash \exists f : \alpha \rightarrow \beta.(\lambda x.fx) = f$
4.	SELECT_AX
	$\vdash \exists P : \alpha \rightarrow bool.Px \Rightarrow P(\epsilon P)$
5.	INFINITY_AX
	$\vdash \exists f : ind \rightarrow ind.\mathrm{One\_One} f \wedge \neg(\mathrm{Onto} f)$

Table 6: HOL core inference rules

1.	Assumption Introduction
	ASSUME: $\frac{}{\{P\} \vdash P}$
2.	Reflexivity
	REFL: $\frac{}{\vdash N = N}$
3.	Beta Conversion
	BETA_CONV: $\frac{}{\vdash (\lambda v.N)M = N[M/v]}$
4.	Abstraction
	ABS: $\frac{\Gamma \vdash M = N}{\Gamma \vdash (\lambda v.M) = (\lambda v.N)}$ (v not free in Γ)
5.	Type Instantiation
	INST_TYPE: $\frac{\Gamma \vdash P}{\Gamma \vdash P[\sigma_1, ..., \sigma_n / \alpha_1, ..., \alpha_n]}$
6.	Discharging Assumption:
	DISCH: $\frac{\Gamma \vdash P}{\Gamma - \{Q\} \vdash Q \supset P}$
7.	Modus Ponens
	MP: $\frac{\Gamma_1 \vdash P \supset Q \quad \Gamma_2 \vdash P}{\Gamma_1 \cup \Gamma_2 \vdash Q}$
8.	Substitution
	SUBST: $\frac{\Gamma_1 \vdash N_1' \quad ... \quad \Gamma_n \vdash N_n = N_n' \quad \Gamma \vdash P}{\Gamma_1 \cup ... \cup \Gamma_n \cup \Gamma \vdash P[N_1', ..., N_n' / N_1, ..., N_n]}$

Table 7: HOL notation

HOL Notation	Standard Notation	Meaning
T	$\top$, true	true
F	$\bot$, false	false
t	$\neg t$	not t
$t_1 \backslash / t_2$	$t_1 \vee t_2$	t_1 or t_2
$t_1 / \backslash\, t_2$	$t_1 \wedge t_2$	t_1 and t_2
$t_1 ==> t_2$	$t_1 \Rightarrow t_2, t_1 \supset t_2$	t_1 implies t_2
$t_1 = t_2$	$t_1 = t_2$	t_1 equals t_2
$t_1 = t_2$	$t_1 \equiv t_2$	t_1 equivalent to t_2
$\backslash x.t$	$\lambda x.t$	lambda function notation
$!x.t$	$\forall x.t$	t holds for all x
$?x.t$	$\exists x.t$	t holds for some x
$?!x.t$	$\exists !x.t$	t holds for precisely one x
$@x.t$	$\epsilon x.t$	an x for which t holds
if t_1 then t_2 else t_3	$t_1 \rightarrow t_2 \mid t_3$	if t_1 then t_2 else t_3
$t_1 > t_2$	$t_1 > t_2$	t_1 is greater than t_2
$t_1 >= t_2$	$t_1 \geq t_2$	t_1 is greater or equal than t_2
$t_1 < t_2$	$t_1 < t_2$	t_1 is less than t_2
$t_1 <= t_2$	$t_1 \leq t_2$	t_1 is less or equal than t_2

Another way to do the same thing is to type the string we want to parse between the special quotation marks (--` and `--). We can enter the term as follows:

```
- (--`A /\ B`--);
> val it = ``A /\ B`` : term
```

Terms in the HOL logic are represented by the ML datatype term. The HOL logic is also typed. The term we just entered was a boolean. The types of the HOL logic are represented by another ML datatype called hol_type. The function type_of: term -> hol_type will tell you the HOL type of a term.

The HOL logic can be conservatively extended with new types and new constants. The simplest way to add a new constant c is to give a definition of the form $c = t$ where t is a closed term (a term without free variables). An extension by constant definition is always a conservative extension, i.e., it is guaranteed not to introduce inconsistencies.

The ML function used to define a new function is new_definition:(string * term) -> thm. For example, a tripling operation can be introduced on the natural numbers by evaluating:

```
new_definition("triple_DEF",(--`tpl = \x. x + x + x`--));
```

The constant definition facility also allows arguments to be given on the left hand side; we could have written:

```
new_definition("triple_DEF",(--`tpl x = x + x + x`--));
```

This adds the constant `tpl:num->num` to the logic and stores the definition in the current theory file under the name `triple_DEF`. Note that this does not bind the definition to a name in the current environment (actually, it is bound to the name `it`). If we want to bind the definition to the name `triple_DEF`, then we should evaluate:

```
val triple_DEF =
  new_definition("triple_DEF",(--`tpl x = x + x + x`--));
```

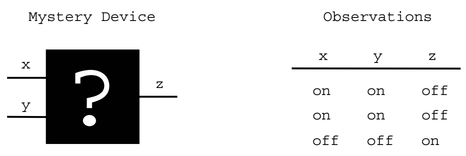
Now suppose we have already decided what goal we would like to prove in HOL and started a proving process by typing `set_goal` command. What would be the best strategy to attack the goal? A very general scheme would be the following:[15]

1. Check whether it is possible to prove (or at least simplify) your goal using existing HOL theorems;
2. If not, expand definitions of all (or some) constants in the goal conclusion;
3. Simplify the goal conclusion (by using beta conversion, removing quantifiers, splitting the goal into simpler subgoals, moving a part of the goal conclusion into the goal assumptions, doing boolean case analysis, ...);
4. Check whether it is possible to prove (or at least simplify) the goal conclusion by rewriting it with trivial rewrites (`REWRITE_TAC []`) and/or the goal assumptions (`ASM_REWRITE_TAC []`);
5. If a goal is still not proved, repeat the procedure starting from the step 1.

4.2 Hardware verification using HOL

The hardware verification process in HOL usually has three steps:[11]

1. Describe the specification
2. Describe the implementation
3. Prove that the implementation meets the specification



x	y	z
on	on	off
on	on	off
off	off	on

Figure 6: Hardware model in HOL

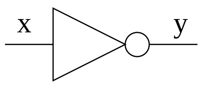


Figure 7: NOT gate

The first step in the verification of hardware is to write a formal specification of the required behavior for the design in HOL. How do we describe a device? The general approach is to model it as a black box in Figure 6. We neglect detailed infrastructure inside the box and only concentrate on its response to the environment outside the box.

Observations of this mystery device can help us to describe hardware in HOL logic:

- Wires can have the value on or off. We model them with boolean variables.
- Devices constrain the values that can be observed on the attached wires. We model these with predicates on wires.

Following this approach, it is possible to express any combinatorial circuit with NOT (Figure 7), AND (Figure 8) and OR (Figure 9) gates, as well as with some means for a line to be tied HI or LO (Figure 10).

```
val NOT_DEF =
  new_definition("NOT_DEF",(--‘NOT x x’ =
  (x’ = ~x)‘--));
val AND_DEF =
  new_definition("AND_DEF",(--‘AND (x,y) x’ =
  (x’ = (x /\ y))‘--));
val OR_DEF  =
  new_definition("OR_DEF",(--‘OR (x,y) x’ =
  (x’ = (x \/ y))‘--));
```

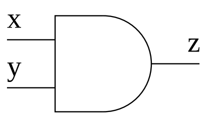


Figure 8: AND gate

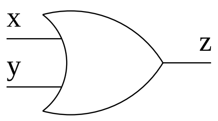


Figure 9: OR gate

Figure 10: Power and ground

```
val HI_DEF  =
  new_definition("HI_DEF",(--`HI x = (x = T)`--));
val LO_DEF  =
  new_definition("LO_DEF",(--`LO x = (x = F)`--));
```

In practice, it is possible to construct any combinatorial circuit purely from NAND (Figure 11) gates or purely from NOR (Figure 12) gates. And, since it is easier to fabricate circuits that only use one kind of gates, this is what is actually done in industrial practice.

```
val NAND_DEF =
  new_definition("NAND_DEF",(--`NAND (x,y) x' =
   (x' = ~(x /\ y))`--));
val NOR_DEF  =
  new_definition("NOR_DEF",(--`NOR (x,y) x' =
   (x' = ~(x \/ y))`--));
```

For example, the implementation of a OR gate by using only NAND gates (Figure 13) can be defined in HOL as below:

```
val OR_IMP = new_definition("OR_IMP",
   (--`OR_IMP (x, y) x' = (? a b c d.
     (HI a) /\ (HI b) /\ (NAND (x, a) c) /\
     (NAND (y, b) d) /\ (NAND (c, d) x'))`--));
```

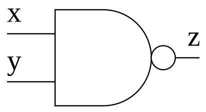


Figure 11: NAND gate

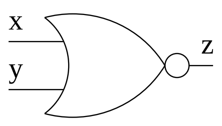


Figure 12: NOR gate

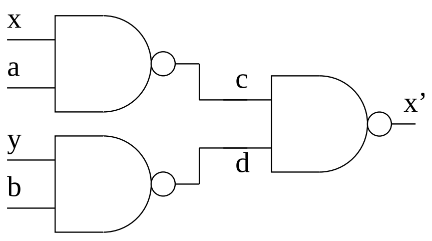


Figure 13: Implementation of OR gate by using only NAND gates

Hereby we can do the verification of the circuit. We would like to know that our design for an OR gate in terms of NAND gates actually functions as an OR gate is supposed to. To establish this fact, we must do the following:

1. Begin the proof by rewriting with definitions.

```
- set_goal([], (--`!x y x'. OR_IMP (x, y) x' ==>
  OR (x, y) x'`--));
> val it =
    Proof manager status: 1 proof.
    1. Incomplete:
         Initial goal:
         !x y x'. OR_IMP (x,y) x' ==> OR (x,y) x'

     : proofs
- e(REWRITE_TAC[OR_IMP, OR_DEF]);
OK..
1 subgoal:
> val it =
    !x y x'.
      (?a b c d.
       HI a /\ HI b /\ NAND (x,a) c /\
       NAND (y,b) d /\ NAND (c,d) x') ==>
       (x' = x \/ y)

     : goalstack
- e(REWRITE_TAC[HI_DEF, NAND_DEF]);
OK..
```

```
1 subgoal:
> val it =
    !x y x'.
      (?a b c d.
         a /\ b /\ (c = ~(x /\ a)) /\ (d = ~(y /\ b)) /\
        (x' = ~(c /\ d))) ==> (x' = x \/ y)

     : goalstack
```

2. The next step is to strip the goal down to its simplest form.

```
- e(REPEAT STRIP_TAC);
OK..
1 subgoal:
> val it =
    x' = x \/ y
    ------------------------------------
      0.  a
      1.  b
      2.  c = ~(x /\ a)
      3.  d = ~(y /\ b)
      4.  x' = ~(c /\ d)
     : goalstack
```

3. To prove the goal, we may need to use De Morgans Law. [1]

```
- e(ASM_REWRITE_TAC[DE_MORGAN_THM]);
OK..

Goal proved.
 [.....] |- x' = x \/ y

Goal proved.
|- !x y x'.
     (?a b c d.
      a /\ b /\ (c = ~(x /\ a)) /\ (d = ~(y /\
      b)) /\ (x' = ~(c /\ d))) ==> (x' = x \/ y)

Goal proved.
|- !x y x'.
     (?a b c d.
```

[1]Another approach is to use boolean cases analysis. This is the theorem proving equivalent of using truth tables. The tactic is called BOOL_CASES_TAC.

```
          HI a /\ HI b /\ NAND (x,a) c /\
          NAND (y,b) d /\ NAND (c,d) x') ==>
          (x' = x \/ y)
    > val it =
        Initial goal proved.
        |- !x y x'. OR_IMP (x,y) x' ==>
        OR (x,y) x' : goalstack
```

4.3 LED case study

In order to make the verification process simpler, we use a step-wise approach to the whole case. First we prove that the schematic design (Figure 4) satisfies our original description (Table 4). Then we prove that the circuit design (Figure 5) meets the requirements of the schematic design.

The specification of each component and thus the whole schematic design is shown below:

```
val NOT_DEF =
  new_definition("NOT_DEF",
    (--`NOT a x = (x = ~a)`--));
val NAND_DEF =
  new_definition("NAND_DEF",
    (--`NAND a b x = (x = ~(a /\ b))`--));
val NAND3_DEF =
  new_definition("NAND3_DEF",
    (--`NAND3 a b c x = (x = ~(a /\ b /\ c))`--));
val NAND7_DEF =
  new_definition("NAND7_DEF",
    (--`NAND7 a b c d e f g x =
      (x = ~(a /\ b /\ c /\ d /\ e /\ f /\ g))`--));

val LED_A_DEF =
  new_definition("LED_A_DEF",
    (--`LED_A_DEF w x y z a =
      (a = if ((w = F) /\ (x = F) /\ (y = F) /\
      (z = T)) \/
      ((w = F) /\ (x = T) /\ (y = F) /\ (z = F)) \/
      ((w = T) /\ (x = F) /\ (y = T) /\ (z = T)) \/
      ((w = T) /\ (x = T) /\ (y = F) /\ (z = T))
      then F else T)`--));

val LED_A_IMP =
  new_definition("LED_A_IMP",
    (--`LED_A_IMP w x y z a =
     ?tw tx ty tz t1 t2 t3 t4 t5 t6 t7.
     (NOT w tw) /\ (NOT x tx) /\ (NOT y ty) /\
     (NOT z tz) /\ (NAND y tz t1) /\ (NAND tx tz t2) /\
     (NAND tw y t3) /\ (NAND x y t4) /\ (NAND w tz t5)
      /\ (NAND3 tw x z t6) /\ (NAND3 w tx ty t7) /\
     (NAND7 t1 t2 t3 t4 t5 t6 t7 a)`--));
```

To facilitate proving process, we try to use several high-level automation tools in the HOL system which allow us to automatically prove or substantially simplify some logical formulas. The most popular automation tools are Simplifier (`simpLib`), Decision Procedures (`decisionLib`), and First-order Prover (`mesonLib`). These three libraries together with some other helpful functions are incorporated into one big library - `bossLib`. With the help of high-level automation tools, the proof length is greatly reduced.

```
- load "bossLib";
- load "simpLib";
- load "mesonLib";
- open bossLib;
- open simpLib;
- open mesonLib;
```

Hereby we can carry out the verification process:

1. Begin the proof by rewriting with definitions.

```
- set_goal([],(--`!w x y z a.
  LED_A_IMP w x y z a ==> LED_A_DEF w x y z a`--));
> val it =
    Proof manager status: 1 proof.
    1. Incomplete:
         Initial goal:
         !w x y z a. LED_A_IMP w x y z a ==>
          LED_A_DEF w x y z a

      : proofs
- e(REWRITE_TAC[LED_A_IMP, LED_A_DEF]);
OK..
1 subgoal:
> val it =
    !w x y z a.
      (?tw tx ty tz t1 t2 t3 t4 t5 t6 t7.
         NOT w tw /\ NOT x tx /\ NOT y ty /\
         NOT z tz /\ NAND y tz t1 /\
         NAND tx tz t2 /\ NAND tw y t3 /\
         NAND x y t4 /\ NAND w tz t5 /\
         NAND3 tw x z t6 /\ NAND3 w tx ty t7 /\
         NAND7 t1 t2 t3 t4 t5 t6 t7 a) ==>
      (a =
       (if
          ~w /\ ~x /\ ~y /\ z \/ ~w /\ x /\ ~y
          /\ ~z \/ w /\ ~x /\ y /\ z \/ w /\ x
          /\ ~y /\ z
        then
          F
        else
          T))

     : goalstack
- e(REWRITE_TAC[NOT_DEF, NAND_DEF,
    NAND3_DEF, NAND7_DEF]);
OK..
1 subgoal:
> val it =
    !w x y z a.
      (?tw tx ty tz t1 t2 t3 t4 t5 t6 t7.
         (tw = ~w) /\ (tx = ~x) /\ (ty = ~y) /\
         (tz = ~z) /\ (t1 = ~(y /\ tz)) /\
         (t2 = ~(tx /\ tz)) /\ (t3 = ~(tw /\ y)) /\
         (t4 = ~(x /\ y)) /\ (t5 = ~(w /\ tz)) /\
         (t6 = ~(tw /\ x /\ z)) /\ (t7 = ~(w /\ tx
          /\ ty)) /\ (a = ~(t1 /\ t2 /\ t3 /\ t4 /\
          t5 /\ t6 /\ t7))) ==>
      (a =
       (if
          ~w /\ ~x /\ ~y /\ z \/ ~w /\ x /\ ~y /\ ~z
          \/ w /\ ~x /\ y /\ z \/ w /\ x /\ ~y /\ z
        then
          F
        else
          T))

     : goalstack
```

2. Use Simplifier to simplify the expression.

```
  - e(SIMP_TAC std_ss []);
  OK..
  1 subgoal:
```

```
> val it =
    !w x y z.
     y /\ ~z \/ ~x /\ ~z \/ ~w /\ y \/ x /\ y \/
     w /\ ~z \/ ~w /\ x /\ z \/ w /\ ~x /\ ~y =
     (w \/ x \/ y \/ ~z) /\ (w \/ ~x \/ y \/ z) /\
     (~w \/ x \/ ~y \/ ~z) /\ (~w \/ ~x \/ y \/ ~z)

     : goalstack
```

3. Remove universally quantified variables from the front of the subgoal.

```
- e(REPEAT GEN_TAC);
OK..
1 subgoal:
> val it =
   y /\ ~z \/ ~x /\ ~z \/ ~w /\ y \/ x /\ y \/
   w /\ ~z \/ ~w /\ x /\ z \/ w /\ ~x /\ ~y =
   (w \/ x \/ y \/ ~z) /\ (w \/ ~x \/ y \/ z) /\
   (~w \/ x \/ ~y \/ ~z) /\ (~w \/ ~x \/ y \/ ~z)

     : goalstack
```

4. Use boolean cases analysis and rewrite the results.

```
- e(BOOL_CASES_TAC(--‘w:bool‘--) THEN REWRITE_TAC[]);
OK..
2 subgoals:
> val it =
    y /\ ~z \/ ~x /\ ~z \/ y \/ x
    /\ y \/ x /\ z =
    (x \/ y \/ ~z) /\ (~x \/ y \/ z)


    y /\ ~z \/ ~x /\ ~z \/ x /\ y
    \/ ~z \/ ~x /\ ~y =
    (x \/ ~y \/ ~z) /\ (~x \/ y \/ ~z)

     : goalstack
```

5. Use First-order Prover to prove the goal. (Since we get two subgoals now, we should apply this tactic to both of them.)

```
- e(MESON_TAC[]);
OK..
Meson search level: ......

Goal proved.
|- y /\ ~z \/ ~x /\ ~z \/ x /\ y \/
   ~z \/ ~x /\ ~y = (x \/ ~y \/ ~z)
   /\ (~x \/ y \/ ~z)

Remaining subgoals:
> val it =
    y /\ ~z \/ ~x /\ ~z \/ y \/ x /\
    y \/ x /\ z = (x \/ y \/ ~z) /\
    (~x \/ y \/ z)

     : goalstack
- e(MESON_TAC[]);
OK..
Meson search level: ......

Goal proved.
|- y /\ ~z \/ ~x /\ ~z \/ y \/ x /\
   y \/ x /\ z = (x \/ y \/ ~z) /\
   (~x \/ y \/ z)

Goal proved.
|- y /\ ~z \/ ~x /\ ~z \/ ~w /\ y
   \/ x /\ y \/ w /\ ~z \/ ~w /\ x
   /\ z \/ w /\ ~x /\ ~y =
   (w \/ x \/ y \/ ~z) /\ (w \/ ~x
   \/ y \/ z) /\ (~w \/ x \/ ~y \/
   ~z) /\ (~w \/ ~x \/ y \/ ~z)

Goal proved.
|- !w x y z.
     y /\ ~z \/ ~x /\ ~z \/ ~w /\ y \/
     x /\ y \/ w /\ ~z \/ ~w /\ x /\ z
     \/ w /\ ~x /\ ~y =
     (w \/ x \/ y \/ ~z) /\ (w \/ ~x \/
     y \/ z) /\ (~w \/ x \/ ~y \/ ~z)
     /\ (~w \/ ~x \/ y \/ ~z)

Goal proved.
|- !w x y z a.
```

```
         (?tw tx ty tz t1 t2 t3 t4 t5 t6 t7.
           (tw = ~w) /\ (tx = ~x) /\ (ty = ~y)
            /\ (tz = ~z) /\ (t1 = ~(y /\ tz)) /\
            (t2 = ~(tx /\ tz)) /\ (t3 = ~(tw /\ y))
            /\ (t4 = ~(x /\ y)) /\ (t5 = ~(w /\ tz))
            /\ (t6 = ~(tw /\ x /\ z)) /\ (t7 = ~(w
            /\ tx /\ ty)) /\ (a = ~(t1 /\ t2 /\ t3 /\
            t4 /\ t5 /\ t6 /\ t7))) ==>
         (a =
          (if
             ~w /\ ~x /\ ~y /\ z \/ ~w /\ x /\ ~y
             /\ ~z \/ w /\ ~x /\ y /\ z \/ w /\ x
             /\ ~y /\ z
           then
             F
           else
             T))

    Goal proved.
    |- !w x y z a.
         (?tw tx ty tz t1 t2 t3 t4 t5 t6 t7.
            NOT w tw /\ NOT x tx /\ NOT y ty /\
            NOT z tz /\ NAND y tz t1 /\
            NAND tx tz t2 /\ NAND tw y t3 /\
            NAND x y t4 /\ NAND w tz t5 /\
            NAND3 tw x z t6 /\ NAND3 w tx ty t7 /\
            NAND7 t1 t2 t3 t4 t5 t6 t7 a) ==>
         (a =
          (if
             ~w /\ ~x /\ ~y /\ z \/ ~w /\ x /\ ~y
             /\ ~z \/ w /\ ~x /\ y /\ z \/ w /\ x
             /\ ~y /\ z
           then
             F
           else
             T))
    > val it =
        Initial goal proved.
        |- !w x y z a. LED_A_IMP w x y z a ==>
        LED_A_DEF w x y z a : goalstack
```

The next step is to prove that our circuit design meets all the requirements of our schematic design, where the whole proof is very similar to the above proof. In order to keep the text concise, we don’t list down those proofs.

When the verification task of the first part is completed, we verify the other six parts with the same method. For the same reason, here we don't list down the proofs of the other six parts due to the similarity in these verification processes.

5 Verification in PVS

PVS stands for Prototype Verification System, and as the name suggests, it is a prototype environment for specification and verification. The primary purpose of PVS is to provide formal support for conceptualization and debugging in the early stages of the lifecycle of the design of a hardware or software system. The primary emphasis in the PVS proof checker is on supporting the construction of readable proofs[24]. There are some nice features of PVS which make it a popular verification tool:[25]

- An expressive specification language that augments classical higher order logic with a sophisticated type system containing predicate subtypes, and with parameterized theories and a mechanism for defining abstract datatypes such as lists and trees.

- A powerful interactive theorem prover. The basic deductive steps in PVS are large compared with many other systems: there are atomic commands for induction, quantifier reasoning, automatic conditional rewriting, simplification using arithmetic and equality decision procedures and type information, and propositional simplification using binary decision diagrams. Model checking capabilities used for automatically verifying temporal properties of finite state systems are also integrated into PVS.

- A friendly (but not advanced) user interface which is strongly integrated with Emacs.

5.1 PVS overview

The PVS specification language is built on classical typed higher-order logic with the usual base types `bool, nat, rational, real` among others and the function type constructor `[A -> B]`. A distinctive feature of the PVS specification language is predicate subtyping. A subtype {`x: A | P(x)`} consists of exactly those elements `a` of type `A` satisfying predicate `P(a)`. Predicate subtypes are used to explicitly constrain the domain and ranges of operations in a specification and to define partial functions.

A PVS specification consists of a number of theories. A theory is a collection of declarations: types, constants (including functions), axioms that express properties about the constants, and theorems and lemmas to be proved. Theories may import other theories and may be parametric in types and constants.

A proof goal in PVS is represented by a sequent. PVS differs from most proof checkers in providing primitive inference rules that are quite powerful, which also perform steps such as quantifier instantiation, rewriting, beta-reduction, and boolean simplification. Proofs and partial proofs can be saved, edited, and rerun.

To illustrate the above ideas, we consider a simple example to introduce the PVS system. Suppose the file `sum.pvs`[2] contains:

```
sum: THEORY
 BEGIN

 n: VAR nat

 sum(n): RECURSIVE nat =
  (IF n = 0 THEN 0 ELSE n + sum(n - 1) ENDIF)
  MEASURE id

 closed_form: THEOREM sum(n) = (n * (n + 1))/2

 END sum
```

This specifies a theory called `sum` in which:[18]

1. The variable `n` is declared to have the (predefined) type `nat`;
2. a function `sum` is defined recursively (the well-foundness of the recursion is explicitly justified by the supplied measure - n in this example);
3. a theorem called `closed_form` is conjectured.

If we run PVS on the file `sum.pvs`, an Emacs window containing its contents will pop up. To prove it[3], we type `META-x prove`. This initiates the parsing and typechecking of the theory containing the conjecture. This takes a few seconds and one is then prompted with `Rule?` for a proof command. Responding to it with `(induct "n")` results in the output:[4]

```
Rule? (induct "n")
Inducting on n on formula 1,
this yields  2 subgoals:
closed_form.1 :

  |-------
{1}   sum(0) = 0 * (0 + 1) / 2
```

[2]This file can be found in `./pvs/Examples` directory.

[3]Alternatively, the official proof given by the PVS team can be found in `./pvs/Examples/sum.prf` file.

[4]Alternatively, the proof can be done fully automatically by responding to it with `(induct-and-simplify "n")`.

As in HOL, the subgoals are stacked and the first one is presented to the user, followed by another prompt for a proof command. This goal is solved using PVS proof command `(grind)`. The subgoal is popped and the remaining goal is presented:

```
Rule? (grind)
sum rewrites sum(0)
  to 0
Trying repeated skolemization, instantiation,
and if-lifting,

This completes the proof of closed_form.1.

closed_form.2 :

  |-------
{1}   FORALL j:
        sum(j) = j * (j + 1) / 2 IMPLIES
         sum(j + 1) = (j + 1) * (j + 1 + 1) / 2
```

This is also solved automatically by PVS proof command `(grind)`.

```
Rule? (grind)
sum rewrites sum(1 + j)
  to 1 + j + sum(j)
Trying repeated skolemization, instantiation,
and if-lifting,

This completes the proof of closed_form.2.

Q.E.D.
```

The theory has now been proved, and typing `META-x spt` shows the proof status of the theory:

```
Proof summary for theory sum
   sum_TCC1......proved - complete   [U]( n/a s)
   sum_TCC2......proved - complete   [U]( n/a s)
   closed_form......proved - complete   [O](0.31 s)
   Theory totals: 3 formulas, 3 attempted,
                  3 succeeded (0.31 s)
```

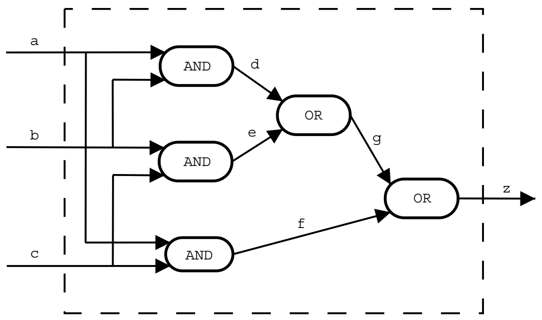


Figure 14: Majority voting circuit[7]

5.2 Hardware verification using PVS

Because the popularity of Gordon's style[19][12] of specifying hardware components in higher order logic, PVS takes the same approach as HOL. The behavior of hardware components is specified by defining predicates that state which combinations of values can appear on their external ports. The behavior of device built by wiring together smaller devices is represented by conjoining the predicates that specify the behaviors of their components with logical conjunction and using existential quantification to hide internal signals.

We illustrate PVS approach by showing a small example of the verification of majority voting circuit[6][7] in PVS. The circuit in Figure 14 is a simplified version of a majority voting circuit as found in nuclear reactors or avionics where three computers each do the same task. If at least two computers signal to do the same thing (i.e. at least two of a, b and c are high) then z is high and the task is performed; otherwise z is low and the task is not performed.[26][23]

We first write the *specification* that asserts the right relationships between inputs (a, b and c) and output (z). The specification is written in a way that is free of implementation detail, and we will not describe any AND/OR gates, just the relationship that ought to hold between the inputs and the outputs. We then write the *implementation* in terms of the AND/OR gates. Finally, we must prove that: *implementation* $\Rightarrow$ *specification*.

The *specification* and *implementation* written in the PVS description language are shown below:

```
major_vote: THEORY

  BEGIN

  % input and output variables
  a, b, c, z: VAR bool

  % conversion function
  cnf(x: bool): int =
10      (IF x THEN 1 ELSE 0 ENDIF)
11
12    % specify the required behavior
13    spec(a, b, c, z): bool =
14      z = (cnf(a) + cnf(b) + cnf(c) >= 2)
15
16    % define and_gate
17    and_gate(v, w, x: bool): bool =
18      x IFF (v AND w)
19
20    % define or_gate
21    or_gate(v, w, x: bool): bool =
22      x IFF (v OR w)
23
24    % describe the implementation
25    implementation(a, b, c, z): bool =
26      (EXISTS (d, e, f, g: bool):
27        and_gate(a, b, d)
28        AND and_gate(b, c, e)
29        AND and_gate(c, a, f)
30        AND or_gate(d, e, g)
31        AND or_gate(g, f, z))
32
33    % the result of cnf is either 0 or 1
34    sanity_check_1: THEOREM
35      (FORALL (d: bool): cnf(d) = 0 OR cnf(d) = 1)
36
37    implementation_correctness: THEOREM
38      implementation(a, b, c, z) IMPLIES
        spec(a, b, c, z)
39
40    END major_vote
```

At line 6 we define the boolean variables a, b, c and z. Thus, wherever these variables occur free in the sequel, they will have type $bool$.

In order to write a succinct specification for majority voting, we first define the conversion function *cnf* at line 9 by:

$$\textit{cnf: bool} \rightarrow \textit{int}$$

The function takes an argument of type $bool$ and returns a value of type int. At line 9, the *cnf* function is defined as follows:

$$\textit{cnf(x) = (if x then 1 else 0)}$$

The if/then/else operator takes as its first argument a boolean expression, and as its second and third operator, arguments of type INT. It returns a value of type int. With the help of the conversion function, lines 13 and 14 define *specification* as being a boolean expression in the input and output variables as shown.

We now want to see if we can implement the specification with hardware gates which are defined at line 17 and 21. The boolean expression $(v\, OR\, w)$ at line 22 is a well-formed formula of the PVS logic, where v and w are boolean variables. "OR" is the PVS notation for standard logical $v \vee w$; the same "OR" symbol is also used in the theorem at line 35.

The $implementation$ in terms of AND/OR gates is described at line 25. Implementation correctness, i.e. *implementation* $\Rightarrow$ *specification* is stated as a theorem to be proved at line 37. The two theorems at lines 34 and 37 are proved automatically in this case:

```
sanity_check_1 :

  |-------
{1}   (FORALL (d: bool): cnf(d) = 0 OR cnf(d) = 1)

Rule? (grind)
cnf rewrites cnf(d)
  to (IF d THEN 1 ELSE 0 ENDIF)
Trying repeated skolemization, instantiation,
and if-lifting,
Q.E.D.


implementation_correctness :

  |-------
{1}   FORALL (a, b, c, z: bool):
        implementation(a, b, c, z) IMPLIES
        spec(a, b, c, z)

Rule? (grind)
and_gate rewrites and_gate(a, b, d)
  to d IFF (a AND b)
and_gate rewrites and_gate(b, c, e)
  to e IFF (b AND c)
and_gate rewrites and_gate(c, a, f)
  to f IFF (c AND a)
or_gate rewrites or_gate(d, e, g)
  to g IFF (d OR e)
or_gate rewrites or_gate(g, f, z)
```

```
  to z IFF (g OR f)
implementation rewrites
implementation(a, b, c, z)
  to EXISTS (d, e, f, g: bool):
             d IFF (a AND b) AND e IFF (b AND c)
             AND f IFF (c AND a) AND g IFF (d OR e)
             AND z IFF (g OR f)
cnf rewrites cnf(a)
  to (IF a THEN 1 ELSE 0 ENDIF)
cnf rewrites cnf(b)
  to (IF b THEN 1 ELSE 0 ENDIF)
cnf rewrites cnf(c)
  to (IF c THEN 1 ELSE 0 ENDIF)
spec rewrites spec(a, b, c, z)
  to z =
       ((IF a THEN 1 ELSE 0 ENDIF) +
        (IF b THEN 1 ELSE 0 ENDIF) +
         (IF c THEN 1 ELSE 0 ENDIF)
         >= 2)
Trying repeated skolemization, instantiation,
and if-lifting,
Q.E.D.
```

5.3 LED case study

Follow the same approach as 4.3, first we prove that the schematic design (Figure 4) satisfies our original description (Table 4). Then we prove that the circuit design (Figure 5) meets the requirements of the schematic design. Like 4.3, here we only show the first part of the whole verification.

The schematic components and connections are modeled in PVS[10] as below:

```
logic_gates: THEORY

  BEGIN

  % input and output
  W, X, Y, Z, a: VAR bool

  % define not_gate
  not_gate(i, j: bool): bool =
    j = NOT i

  % define 2 input nand_gate
  nand_gate2(i, j, k: bool): bool =
    k = NOT (i AND j)

  % define 3 input nand_gate
  nand_gate3(i0, i1, i2, j: bool): bool =
    j = NOT (i0 AND i1 AND i2)

  % define 7 input nand_gate
  nand_gate7(i0, i1, i2, i3, i4, i5, i6, j: bool)
    : bool = j = NOT (i0 AND i1 AND i2 AND i3 AND
                      i4 AND i5 AND i6)

  % specification
  spec(W, X, Y, Z, a): bool =
    NOT a = (W = FALSE AND X = FALSE AND Y = FALSE
             AND Z = TRUE) OR (W = FALSE AND X = TRUE
             AND Y = FALSE AND Z = FALSE) OR
            (W = TRUE AND X = FALSE AND Y = TRUE AND
             Z = TRUE) OR (W = TRUE AND X = TRUE AND
             Y = FALSE AND Z = TRUE)

  % implementation
  imp(W, X, Y, Z, a): bool =
    (EXISTS (tw, tx, ty, tz, t1, t2, t3, t4, t5,
     t6, t7: bool):
       not_gate(W, tw) AND not_gate(X, tx) AND
       not_gate(Y, ty) AND not_gate(Z, tz) AND
       nand_gate2(Y, tz, t1) AND nand_gate2(tx,
       tz, t2) AND nand_gate2(tw, Y, t3) AND
       nand_gate2(X, Y, t4) AND nand_gate2(W,
       tz, t5) AND nand_gate3(tw, X, Z, t6) AND
       nand_gate3(W, tx, ty, t7) AND nand_gate7
       (t1, t2, t3, t4, t5, t6, t7, a))

  implementation_correctness: THEOREM
    imp(W, X, Y, Z, a) IMPLIES spec(W, X, Y, Z, a)


  END logic_gates
```

The proof is automatically done with PVS proof command `(grind)`:

```
implementation_correctness :

  |-------
{1}   FORALL (W, X, Y, Z, a: bool):
        imp(W, X, Y, Z, a) IMPLIES
        spec(W, X, Y, Z, a)

Rule? (grind)
not_gate rewrites not_gate(W, tw)
  to tw = NOT W
not_gate rewrites not_gate(X, tx)
  to tx = NOT X
not_gate rewrites not_gate(Y, ty)
  to ty = NOT Y
not_gate rewrites not_gate(Z, tz)
  to tz = NOT Z
nand_gate2 rewrites nand_gate2(Y, tz, t1)
  to t1 = NOT (Y AND tz)
nand_gate2 rewrites nand_gate2(tx, tz, t2)
  to t2 = NOT (tx AND tz)
nand_gate2 rewrites nand_gate2(tw, Y, t3)
  to t3 = NOT (tw AND Y)
nand_gate2 rewrites nand_gate2(X, Y, t4)
  to t4 = NOT (X AND Y)
nand_gate2 rewrites nand_gate2(W, tz, t5)
  to t5 = NOT (W AND tz)
nand_gate3 rewrites nand_gate3(tw, X, Z, t6)
  to t6 = NOT (tw AND X AND Z)
nand_gate3 rewrites nand_gate3(W, tx, ty, t7)
  to t7 = NOT (W AND tx AND ty)
nand_gate7 rewrites nand_gate7(t1, t2, t3, t4,
  t5, t6, t7, a) to a = NOT (t1 AND t2 AND t3
  AND t4 AND t5 AND t6 AND t7)
imp rewrites imp(W, X, Y, Z, a)
  to a = NOT ( NOT (Y AND NOT Z) AND NOT
        (NOT X AND NOT Z) AND NOT (NOT W AND Y)
         AND NOT (X AND Y) AND NOT (W AND NOT Z)
         AND NOT (NOT W AND X AND Z) AND NOT
        (W AND NOT X AND NOT Y))
spec rewrites spec(W, X, Y, Z, a)
  to NOT a = (NOT W AND NOT X AND NOT Y AND Z)
             OR (NOT W AND X AND NOT Y AND NOT Z)
             OR (W AND NOT X AND Y AND Z) OR
             (W AND X AND NOT Y AND Z)
Trying repeated skolemization, instantiation,
and if-lifting,
Q.E.D.
```

When the verification task of the first part is completed, we can verify the other six parts with the same method. In order to keep the text concise, we don't list down the proofs of the other six parts due to the similarity in these verification processes.

6 A Comparison of HOL and PVS

There is an overwhelming number of different proof tools available(e.g. in [4] one can find references to over 60 proof tools). All have particular applications that they are especially suited for. Since we have used HOL and PVS as the mechanical verification tools in the previous chapters, hereby it is desirable to do a comparative study of the two proof tools, because both are known as powerful proof tools for higher order logic, which have shown their capabilities in non-trivial applications.

Generally, although HOL and PVS are similar to each other and shares a lot of common features, partly because they are all based on higher order logic and for supporting formal methods applications with proof, there are still some differences. In this section we wish to discuss in some detail our own, more personal, experiences with regards to the case study:

- The meta-language of HOL is ML; hence HOL type system is similar to the type system of ML, which form the basis of the higher order logic theory. (see 4.1).

  PVS is written in Lisp and implements classical typed higher order logic with an extension of predicate subtypes (see 5.1). PVS has many built-in types and uses type constructors to build complex types.

- The specification language of HOL is a ML-style one, which uses the ML datatype `term` to represent the HOL logic; theories are created in ML functions by `new_definition` (see 4.1).

  ```
  val NOT_DEF =
    new_definition("NOT_DEF",
      (--`NOT a x = (x = ~a)`--));
  ```

  Take a look into the case study in 4.3, we can see that the specification consists of the hardware components specification, the target hardware device specification composed with above components' specification, (and the correctness relationship to be proved by `set_goal`, which looks like a part of the proof).

  ```
  set_goal([],(--`!w x y z a.
    LED_A_IMP w x y z a ==> LED_A_DEF w x y z a`--));
  ```

The specification language of PVS is rich, containing many different type constructors and predicate subtypes (see 5.1). Unlike HOL, the syntax is more fixed; many language constructs, such as `IF` and `CASES` are built-in to the language. A specification is usually divided in several theories and theories can import other theories. Although from the case study in 5.3, we can find out that the specification is organized similarly to 4.3, there are two obvious differences:

- Variables have to be declared before using (there is no default datatype mechanism for undefined variables).

  ```
  % input and output
  W, X, Y, Z, a: VAR bool
  ```

- The correctness relationship to be proved is within `THEORY`.

  ```
  logic_gates: THEORY

  ⋮

  implementation_correctness: THEOREM
  imp(W, X, Y, Z, a) IMPLIES spec(W, X, Y, Z, a)

  END logic_gates
  ```

• HOL supports both forward and backward proving, but it emphasizes on backward proving by supplying many useful tactics for it. A tactic transforms the proof goal into several subgoals (see 4.2). HOL has a large collection of tactics as well as many proving tools. In the process of proving 4.3, we need to load such tools from libraries by `load` before proving because they don't automatically "stand forward" when applicable.

```
load "bossLib";
load "simpLib";
load "mesonLib";
```

A thorough look of HOL libraries beforehand will help us to get familiar with some of powerful proving tools.

PVS has many tools in the core system which can be automatically invoked (see 5.2). We are quite impressed in the process of proving 5.3; such tools are built-in to the system and are ready to use by invoking `grind` etc.

```
implementation_correctness :

|-------
```

```
{1}FORALL (W, X, Y, Z, a: bool):
   imp(W, X, Y, Z, a) IMPLIES spec(W, X, Y, Z, a)

Rule? (grind)
```

Another difference is that after supplying a tactic, the system repeatedly apply it to the current goal until no changes in the current state. A PVS tactic is like a `REPEAT` HOL tactic in this way.

```
e(REPEAT GEN_TAC);
```

- The most famous difference between HOL and PVS is that the former is a LCF-style prover, which has better security, user extensibility and also ways to import and export proofs to other provers.

When comparing HOL and PVS we realized that both tools had their advantages and disadvantages. If we want to built our own ideal proof tool, it should combine the best of both worlds: [8][29][32]

The logic Predicate subtyping gives so much extra expressiveness and protection against semantical errors, that this should be supported.

The specification language The specification language should be readable, expressive and easily extendible. For function application, we have a slight preference for the bracketless syntax of HOL.

The prover The ideal prover has powerful proof commands for classical reasoning and rewriting, including ordered rewriting. A tactic should return a list of possible next states, as this is useful to try all possible instantiations. Also, decision procedures should be available. Preferably, these decision procedures are not built in to the kernel, but written in the tactical language, so that they can not cause soundness problems. The style of the interactive proof commands of PVS is preferred over that of HOL, because this is more intuitive.

System organization To ensure soundness of the proof tool, the system should have a small kernel. The code of the tool should be freely available, so that users can easily extend it for their own purpose and implement bug fixes.

The proof manager and user interface The tool should keep track of the proof trace. Proofs are best represented as trees, because this is more natural, compared to a linear structure. The tree representation also allows easy navigation through the proof, supported by a visual representation of the tree.

7 Concluding remarks and future work

The paper began with an overview on hardware verification methods, with the emphasis on approaches using higher order logic. We selected two popular verification tools, HOL and PVS, and started with some well-understood, but non-trivial examples, then smoothly moved to a practical verification case study of a seven-segment LED display decoder circuit design.

When applying these two tools to our case, we found PVS was easier to use probably because of some “engineering philosophy” in it. However, we also found that PVS was not an open system, which makes it unsuitable for certain kinds of work requiring more flexibilities. Besides, we also found that there were many opportunities for future work in this case study:

- When writing this paper, I found that today the formal verification community suffers from a lack of meaningful and widely distributed examples for evaluating the performance of verification tools. Existing examples in the area of theorem proving are either toyish or trivial. More realistic hardware examples have little documentation and few property specifications. The benefits of a set of examples are many. It will motivate the development of new algorithms. It will also facilitate comparisons across tools and provide case studies of verification methodologies for users.

- In my opinion, it should be possible to simultaneously ensure the secure extensibility of HOL and the usability and power of PVS. One possible hypopaper is to implement a PVS-style proof environment in HOL.

- Both tools lack a user-friendly interface. PVS is strongly integrated with Emacs. The *de facto* interface for HOL is `hol-mode` (also based on Emacs). There are some more advanced user interfaces based on Tcl/Tk, but they only work for particular versions of HOL.

Over the last two decades hardware verification has moved from academic research to a rapidly growing commercial technology.[16] In the past, verification methods have divided into two well-established approaches: theorem proving and model checking.[9] We focus on theorem proving approach in the whole paper. Model checking is a technique that relies on building a finite model of a system and checking that a desired property holds in that model.

In contrast to theorem proving, model checking is completely automatic and fast, sometimes producing an answer in a matter of minutes. The main disadvantage of model checking is the state explosion problem.

Theorem proving can deal directly with infinite state space. It relies on techniques like structural induction to prove over infinite domains, but theorem provers usually require interaction with a human so that the theorem proving process is slow and often error-prone.

One of the most promising directions in hardware verification is combining model checking and theorem proving, ideally to benefit from the advantages of both approaches. One way is to employ model checking as a decision procedure within a deductive framework, as is partly done in tools such as HOL and PVS; another way is to use deduction to obtain a finite state abstraction of an implementation that can be verified using model checking.

Another promising direction in hardware verification is to make specification methods and tools more user-friendly. Although industry is adopting techniques like model checking and theorem proving to complement the more traditional one of simulation, there are still a lot of problems for industry applications. (i.e. The notations are too obscure, and the tool is too hard to use.) Ideally, people from industry expect to use the formal hardware specification language as simply a means of communicating ideas to others or of documenting their own designs. They would use tools like model and proof checkers with as much ease as they use compilers. Therefore, we should strive to make our notations and tools accessible to non-experts.

References

[1] A.Gupta. *Formal Hardware Verification Methods: A Survey*. Journal of Formal Methods in System Design, vol. 1, pp. 151 - 238, 1992

[2] C.Max. *Bebop to the Boolean Boogie*. LLH Technology Publishing, 1997

[3] C.-J.H.Seger. *An Introduction to Formal Hardware Verification*. Technical Report, University of British Columbia, Computer Science Department, Number TR-92-13, 1992

[4] Database of Existing Mechanized Reasoning Systems: available at http://www-formal.stanford.edu/clt/ARS/systems.html

[5] D.D.Gajski, F.Vahid, S.Narayan, and J.Gong. *Specification and Design of Embedded System*. Prentice Hall, 1994

[6] D.Gries, F.B.Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag, 1993

[7] D.Gries, F.B.Schneider. *Equational Propositional Logic*. available at http://www.ariel.cs.yorku.ca/ logicE/

[8] D.Griffioen, M.Huisman. *A Comparison of PVS and Isabelle/HOL*. Theorem Proving in Higher Order Logics: 11th International Conference, vol. 1479, pp. 123 - 142, Springer, 1998

[9] E.Clarke, J.Wing. *Formal Methods: State of the Art and Future Directions*. CMU Computer Science Technical Report, CMU-CS-96-178, 1996

[10] G.C.Gopalakrishnan. *An Overview of Formal Mathematical Reasoning with Applications to Digital System Verification*. available at http://www.cs.utah.edu/formal_verification

[11] J.Grundy. COMP8033: Mechanical Verification Web Site: http://cs.anu.edu.au/student/comp8033/

[12] J.J.Joyce. *More Reasons Why Higher-Order Logic is a Good Formalism for Specifying and Verifying Hardware*. International Workshop on Formal Methods in VLSI Design, 1991.

[13] J.M.Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 2002

[14] KarnaughMap v1.2: available at http://www.puz.com/sw/karnaugh/karnaugh_12.htm

[15] L.Laibinis. Mechanical Verification Course Web Site: http://www.abo.fi/ linas.laibinis/MechVer/MechVer.html

[16] M.Gordon. *21 Years of Hardware Verification*. Talk given at the Royal Society, 1998. available at http://www.cl.cam.ac.uk/ mjcg/BDD/facs21-talk.ps.gz

[17] M.Gordon. *HOL: A Machine Oriented Formulation of Higher Order Logic*. Technical Report 68, Computer Laboratory, University of Cambridge, 1985

[18] M.Gordon. *Notes on PVS from a HOL perspective*. available at http://www.cl.cam.ac.uk/users/mjcg/pvs.ps.gz

[19] M.Gordon. *Why higher-order logic is a good formalism for specifying and verifying hardware*. Formal Aspects of VLSI Design, pp. 153 - 177, North-Holland, 1986

[20] MichiganTech Web Site: http://www.ee.mtu.edu/faculty/schulz/lab_courses/EE2301_fall00/pages/week_4_bcd_to_seven_segment.html

[21] M.John, S.Smith. *Application-Specific Integrated Circuits*. Addison-Wesley, 1997

[22] M.Srivas, H.Rue, D.Cyrluk. *Hardware Verification Using PVS*. Formal Hardware Verification - Methods and Systems in Comparison, Lecture Notes in Computer Science, vol. 1287, pp. 156 - 205, Springer, 1997

[23] N.Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1996.

[24] N.Shankar, S.Owre, J.M.Rushby, D.W.J.Stringer-Calvert. *PVS System Guide*. available at http://pvs.csl.sri.com/doc/pvs-system-guide.pdf

[25] N.Shankar, S.Owre, J.M.Rushby, D.W.J.Stringer-Calvert. *PVS Prover Guide*. available at http://pvs.csl.sri.com/doc/pvs-prover-guide.pdf

[26] N.Storey. *Safety Critical Computer Systems*. Addison-Wesley, 1996.

[27] R.J.Baker, H.W.Li, D.Boyce. *CMOS: Circuit Design, Layout, and Simulation*. John Wiley and Sons publishers, 1998

[28] S.S.Skiena *The Algorithm Design Manual*. Springer-Verlag, 1997

[29] S.Tahar, P.Curzon, and J.Lu. *Three Approaches to Hardware Verification: HOL, MDG and VIS Compared*. Formal Methods in Computer-Aided Design, Lecture Notes in Computer Science, vol. 1522, pp. 433 - 450, Springer, 1998

[30] Tokyo Denki University Web Site: http://www.d.dendai.ac.jp/vhdl/decoder.html

[31] T.Melham. *Higher Order Logic And Hardware Verification*. Cambridge University Press, 1993

[32] V.Zammit. *A Comparative Study of Coq and HOL*. Proceedings of the 10th International Workshop on Theorem Proving in Higher Order Logic. vol. 1275, pp. 323 - 337, Springer, 1997

[33] Windows LASI: layout system for Windows. available at http://members.aol.com/lasicad/index.htm

ISBN 951-29-4020-5
ISSN 1239-1883