Lecture Notes in Computer Science 2060
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

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

Thomas Böhme Herwig Unger (Eds.)

Innovative Internet Computing Systems

International Workshop IICS 2001
Ilmenau, Germany, June 21-22, 2001
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Thomas Böhme
Technische Universität Ilmenau
Institut für Mathematik
Postfach 10 05 65, 98684 Ilmenau, Germany
E-mail: tboehme@mathematik.tu-ilmenau.de

Herwig Unger
Universität Rostock
Fachbereich Informatik
Albert-Einstein-Str. 23, 18051 Rostock, Germany
E-mail: hunger@informatik.uni-rostock.de

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Innovative internet computing systems : proceedings / International Workshop IICS 2001, Ilmenau, Germany, June 21 - 22, 2001. Thomas Böhme ; Herwig Unger (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2001.
(Lecture notes in computer science ; Vol. 2060)
ISBN 3-540-42275-7

CR Subject Classification (1998): C.2, D.2, F.3, H.3-4, I.2.11, K.4.3-4

ISSN 0302-9743
ISBN 3-540-42275-7 Springer-Verlag Berlin Heidelberg New York

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

http://www.springer.de

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN 10781593 06/3142 5 4 3 2 1 0

Preface

Nowadays, the Internet is the most commonly used medium for the exchange of data in different forms. Presently, over 60 million machines have access to the Internet and to its resources. However, the Internet is also the largest distributed system offering different computational services and possibilities not only for cluster computing. If the needs of modern mobile computing and multimedia systems are taken into account, it becomes clear that modern methods must ensure an effective development and management of the Internet allowing each user fast access to this huge resource space.

The Innovative Internet Computing Systems workshop is organized by the Gesellschaft für Informatik (GI) in Germany. It intends to be an open meeting point for scientists dealing with different aspects of this complex topic. In contrast to the Distributed Communities on the Web workshops, which can be considered as the roots of I^2CS, special attention is given to fundamental research works and the application of theoretical and formal results in practical implementations.

The 16 papers (2 invited, 9 long, and 5 short) presented at the conference and in the present volume were selected out of 30 contributions. Every submission was fully reviewed by three members of the program committee. The two invited speakers Paul Tarau (University of North Texas, Denton, U.S.A.) and Knut Löscke (PC-Ware Information Technologies AG Leipzig, Germany) presented overview talks on Multicast Protocols for Jinni Agents and eLicensing respectively, from the scientific as well as the industrial point of view.

Furthermore, Jochen Harant from the Ilmenau Technical University gave a tutorial on some aspects of the probabilistic method in Discrete Mathematics, which we think is becoming relevant for computation in large distributed systems.

We would like to thank all those who contributed to this book for their excellent work and their great cooperation. Mrs. Barbara Hamann deserves special gratitude for her great efforts and perfect work concerning the preparation of this volume and the administrative details associated with the workshop. We wish to acknowledge the substantial help provided by our sponsors: Ilmenau Technical University, the University of Rostock, DKV Deutsche Krankenversicherung AG, and 4FriendsOnly.com Internet Technologies AG.

We hope that all participants enjoyed a successful workshop, made a lot of new contacts, held fruitful discussions helping to solve the actual research problems, and had a pleasant stay in Ilmenau. Last but not least we hope to see you again at I^2CS 2002, which will be held in the Rostock area in the north of Germany.

May 2001

Thomas Böhme
Herwig Unger

Organization

I^{2}CS was organized by the Gesellschaft für Informatik (GI) in Germany.

Executive Committee

Thomas Böhme (Chair)
Herwig Unger (Co-chair)
Barbara Hamann

Program Committee

T. Böhme (Chair)
J. Brooke
M. Bui
S. Blyumin
M. Dietzfelbinger
W. Fengler
G. Hipper
A. Mikler
K. Kleese
P. Kropf
M. Kunde
R. Liskowsky
A. Pears
M.A.R. Dantas
A. Rebetzky
D. Reschke
M. Sommer
D. Tavangarian
H. Unger(Co-chair)
T. Ungerer

Sponsoring Institutions

Ilmenau Technical University
University of Rostock
DKV Deutsche Krankenversicherung AG
4FriendsOnly.com Internet Technologies AG

Table of Contents

Workshop Innovative Internet Computing Systems

Multicast Protocols for Jinni Agents 1
Satyam Tyagi, Paul Tarau, and Armin Mikler

Modern Software Engineering Methods
for IP-QoS Resource Pool Management 19
Gerald Eichler, Falk Fünfstück, Fabio Ricciato, Anne Thomas, Charilaos Tsetsekas, and Martin Winter

Adaptive Quality of Service Management Using QoS Proxy
and User Feedback for Wireless Links 31
Dang-Hai Hoang, Dietrich Reschke, and Werner Horn

Sharing Extendible Arrays in a Distributed Environment 41
Tatsuo Tsuji, Hidetatsu Kawahara, Teruhisa Hochin, and Ken Higuchi

Pini – A Jini-Like Plug&Play Technology for the KVM/CLDC 53
Steffen Deter and Karsten Sohr

Discovering Internet Services: Integrating Intelligent Home Automation
Systems to Plug and Play Networks 67
Wolfgang Kastner and Markus Leupold

Reusing Single-User Applications
to Create Multi-user Internet Applications 79
Stephan Lukosch and Jörg Roth

Graph-Theoretic Web Algorithms: An Overview 91
Narsingh Deo and Pankaj Gupta

Prefetching Tiled Internet Data
Using a Neighbor Selection Markov Chain 103
Yoo-Sung Kim, Ki-Chang Kim, and Soo Duk Kim

A General Adaptive Cache Coherency-Replacement Scheme
for Distributed Systems 116
Jose Aguilar and Ernst Leiss

Agents Based Collaborative Framework for B2C Business Model
and Related Services 126
V. Robin Rohit and D. Sampath

Agent-Based Distributed Computing with JMessengers 134
Moritz Gmelin, Jochen Kreuzinger, Matthias Pfeffer, and Theo Ungerer

Agent-Based Wave Computation:
Towards Controlling the Resource Demand 146
Armin R. Mikler and Vivek S. Chokhani

A Design for JTrader, an Internet Trading Service 159
Marcelo d'Amorim and Carlos Ferraz

Computer-Supported Deliberations for Distributed Teams 167
Jacques Lonchamp and Fabrice Muller

Hardware Security Concept for Spontaneous Network Integration
of Mobile Devices 175
Igor Sedov, Marc Haase, Clemens Cap, and Dirk Timmermann

Author Index 183

Multicast Protocols for Jinni Agents

Satyam Tyagi, Paul Tarau, and Armin Mikler

Department of Computer Science
University of North Texas
P.O. Box 311366
Denton, Texas 76203
{tyagi,tarau,mikler}@cs.unt.edu

Abstract. We extend the Jinni Agent Programming Infrastructure with a multicast network transport layer. We show that the resulting protocol emulates client/server exchanges while providing high performance multicasting of remote calls to multiple sites. To ensure that our agent infrastructure runs efficiently independently of router-level multicast support, we also describe a blackboard based algorithm for locating a randomly roaming agent for message delivery. Jinni's orthogonal design, separating blackboards from networking and multi-threading, turns out to be easy to adapt to support a generalization of Jinni's Linda based coordination model. The resulting system is particularly well suited for building large scale, agent based, IP transparent, fault tolerant, tele-teaching and shared virtual reality applications.

Keywords: Mobile Computations, Intelligent Mobile Agents, Logic Programming, Blackboard based Coordination, Networking Software Infrastructure, Multicast, Java3D based Shared Virtual Reality, IP Transparent Communication

1 Introduction

The advent of networked, mobile, ubiquitous computing has brought a number of challenges, which require new ways to deal with increasingly complex patterns of interaction: autonomous, reactive and mobile computational entities are needed to take care of unforeseen problems, to optimize the flow of communication, to offer a simplified, and personalized view to end users. These requirements naturally lead towards the need for *agent programs* with increasingly sophisticated inference capabilities, as well as autonomy and self-reliance. To host them with minimal interference to their main mission, we need a software infrastructure providing a minimal basic ontology - ideally as simple and self contained as the IP packet universe on top of which the Internet is built.

Conveniently encapsulated as *agents*, software artifacts enabled with autonomy, dynamic knowledge exchange and network transparent mobility (as key features) have emerged.

An important number of early software agent applications are described in [3] and, in the context of new generation networking software, in [30,14].

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 1–18, 2001.

Mobile code/mobile computation technologies are pioneered by General Magic's Telescript (see [17] for their Java based *mobile agent* product) and IBM's Java based Aglets [20]. Other mobile agent and mobile object related work illustrate the rapid growth of the field: [24,18,19,21,31,32,25]

Implementation technologies for mobile code are studied in [1]. Early work on the Linda coordination framework [7,8,4] has shown its potential for coordination of multi-agent systems. The logical modeling and planning aspects of computational Multi-Agent systems have been pioneered by [9,10,11,12,13,22,23,33].

Jinni 2000 is a multi-threaded Java based Prolog system with modular, plugin networking layer, designed as an agent programming library, through a combination of Java and Prolog components. It supports mobile agents either directly through a mobile thread abstraction or as a combination of remote predicate calls, client/server components, multiple networking layers and blackboards.

1.1 Overview of the Paper

The paper is divided into 8 sections. Section 2 describes the Jinni's Mobile Agent Architecture briefly and some of Jinni's functionalities and advantages. Section 3 introduces us to the recently developed Multicast layer for Jinni and some of its advantages. Section 4 outlines the Synchronization of Multicast Agents with Blackboards. Section 5 explores some important properties of multicast networks, which are significant for Jinni in particular, and mobile agents and mobile computing in general. In section 6 we describe two applications for Multicast Agents using the Jinni Agent Programming Infrastructure. Section 7 discusses some of the problems and related areas to be explored in future. Finally we conclude with section 8 stating current achievements, ongoing applications and new possibilities.

2 The Jinni Architecture

2.1 Ontology

Jinni is based on simple **Things, Places, Agents** ontology.

Things are Prolog terms (trees containing constants and variables, which can be unified and other compound sub-terms).

Places are processes with at least one server and a blackboard allowing synchronized multi-user Linda and Remote Predicate Call transactions. The blackboard stores Prolog terms, which can be retrieved by agents.

Agents are collections of threads executing various goals at various places. Each thread is mobile, may visit multiple places and may bring back results.

2.2 Basic Features

Jinni is a Prolog interpreter written in Java, which provides an infrastructure for mobile logic programming (Prolog) based agents. It spawns interpreters as

threads over various network sites and each interpreter has its own state. Computation mobility is mapped to data mobility (through use of meta-interpreters, data can be treated as code). Mobile threads can capture first order "AND"-continuations (as "OR" continuations would cause backtracking, which is not a good idea over the network) and resume execution at remote site by fetching code as needed.

Shared blackboards are used for communication and coordination of agents. Jinni has an orthogonal design and separates high level networking operations (supporting remote predicate calls and code mobility), from Linda coordination code. It has various plugins for GUI, different Network layers (in particular the multicast layer described in this paper) and a Java3d interface, which can be plugged in as an extension. Jinni 2000 embeds a fast incremental compiler, which provides Prolog processing within a factor of 5-10 from the fastest C-based implementations around.

For more details on Jinni see [26,27].

2.3 Data, Code and Computation Mobility

While *data* and *code* mobility present no challenge in a Java environment, migrating the state of the computation from one machine or process to another still requires a separate set of tools. Java's remote method invocations (RMI) add transparent control mobility and a (partially) automated form of *object mobility* i.e. integrated code (class) and data (state) mobility.

Mobility of live code is called *computation mobility* [5]. It requires interrupting execution, moving the state of a runtime system (stacks, for instance) from one site to another and then resuming execution. Clearly, for some languages, this can be hard or completely impossible to achieve (C/C++) while in other languages like Java it still requires class specific serialization methods (providing writing and reading of objects to/from byte streams).

Conventional mobile code systems like IBM's Aglets [20] require serialization hints from the programmer and do not implement a fully generic reflective computation mobility infrastructure. Aglets do not provide code mobility as they assume that code is already available at the destination site. In practice this means that the mobile code/mobile computation layer is not really transparent to the programmer.

In contrast, our architecture is based on building an autonomous layer consisting of a reflective interpreter, which provides the equivalent of implicit serialization and supports orthogonal transport mechanisms for data, code and computation state. The key idea is simply that by introducing interpreters spawned as threads by a server at each networked site, *computation mobility* at object-level is mapped to *data mobility* at meta-level in a very straightforward way. A nice consequence is transport independence coming from the uniform representation of data, code and computation state (in practice this means that Corba, RMI, HLA or plain/multicast sockets can be used interchangeably as a transport mechanism).

2.4 Advantages of Computation Mobility

Current agent programming infrastructures use message passing as the key communication mechanism. Existing Concurrent Constraint Logic Programming languages (and multi-paradigm languages like Distributed Oz) support distributed programming and coordination through monotonic stores and shared logical variables.

By contrast, we want to explore the impact of mobile live computations, lightweight multi-engine/multi-threaded script interpreters, blackboard constraints and multicast based coordination on building mobile multi-agent systems. Our distributed Linda blackboards generalize concurrent constraint programming stores by allowing non-monotonic updates of assertional constraints [27]. With our multicast layer we implement transparently replication of shared blackboards distributed over different places.

We refer to [26,15] for more detailed discussion on the advantages of computation mobility.

3 A Multicast Layer for Jinni

3.1 Multicast

Multicast is a technique that allows data, including packet form, to be simultaneously transmitted to a selected set of destinations on a network. Some networks, such as Ethernet, support multicast by allowing a network interface to belong to one or more multicast groups.

2. To transmit identical data simultaneously to a selected set of destinations in a network, usually without obtaining acknowledgment of receipt of the transmission [29].

The key concepts of Multicast Networks are described in [16]. On an Ethernet (Most popular LAN architecture) each message is broadcasted and the machine seeing its own address grabs the message. The multicast packets are also sent in a similar way except that more than one interface picks them up.

Thus the spreading and cloning of agent threads/agents themselves on the whole network or a subset *multicast group* is now a single step operation. This leads to important speed up especially when multiple copies of same code need to be executed at different places (like for parallel searches or for simultaneous display in shared virtual reality applications).

3.2 Multicast in Jinni

Multicasting has various interesting properties, which make it well suited for an agent platform like Jinni. An important advantage of multicasting agents is that, the same code can be run in parallel at the same time in one single operation at different remote sites, retrieving different data available at different sites.

The API : A minimal API consists of two predicates one to run multicast servers, which service requests for multicast clients and second to send multicast requests to these servers:

run_mul_server This joins a multicast group with an address and port. The server now is listening on this port and can receive remote requests for local execution. (When we join a group we are telling the kernel, "I am interested in this multicast group. So, deliver (to any process interested in them, not only to me) any datagram that you see in this network interface with this multicast group in its destination field.")

run_server, which was there for unicast servers is extended to
run_mul_server defaults to **Host = 224.1.1.1, Port = 7001**
run_mul_server(Port) defaults to **Host = 224.1.1.1**
run_mul_server(Host, Port)

Notice that **run_mul_server** has a Host field as well, because it does not run on the local IP but on a multicast address i.e. 224.x.x.x 239.x.x.x

remote_mul_run(G) This is kind of a goal (G), which is multi-casted to the group to be run remotely on all the multicast servers accepting requests for this group.

remote_run(G), which was there for unicast requests is extended to
remote_mul_run(G) which defaults to **Host = 224.1.1.1, Port = 7001**
remote_mul_run(Host, Port, AnswerPattern, Goal, Password, Result)

Our set of multicast servers runs by listening for packets on their group and by responding back on the group's multicast address. Clients listening on this *multicast group* receive these results. An important issue here is that the server should be able to distinguish between a request and a reply, otherwise it would keep responding back to its own replies. This is solved by introducing a simple header distinguishing the two types of messages.

4 Synchronizing Multicast Agents with Blackboard Constraints

The synergy between mobile code and Linda coordination allows an elegant, component wise implementation. *Blackboard operations are implemented only between local threads and their (shared) local blackboard.* If interaction with remote blackboard is needed, the thread simply moves to the place where it is located and proceeds through local interaction. The interesting thing with multicast is that the thread can be multi-casted to a set of places and can interact at all these places locally. This gives an appearance that all these blackboards are one to the members of this multicast groups. For example the **remote_mul_run(mul_all(a(X),Xs))** operation is multi-casted to all servers in

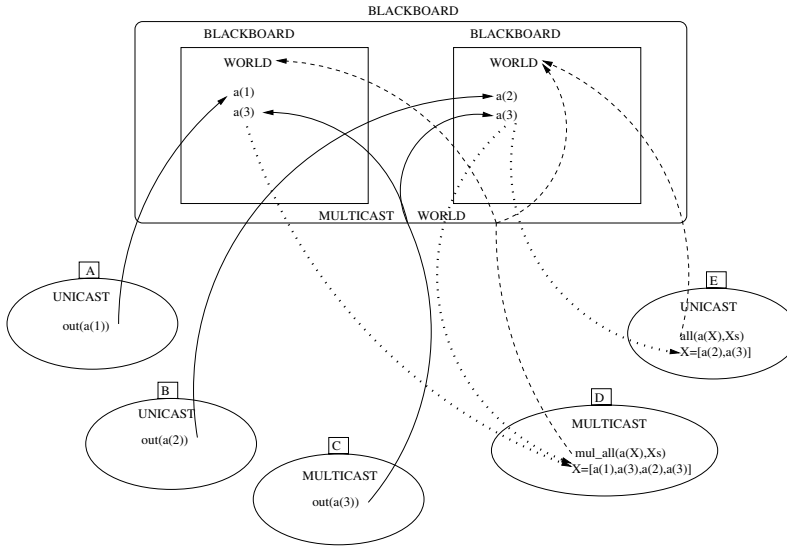


Fig. 1. Basic Linda operations with Multicast

the group. It collects lists of matching results at these remote servers and the output is unicast-ed from all these remote sites to the local blackboard.

This can be achieved as follows:

mul_all(X,Xs):-mul_all('localhost',7001,X,Xs).

mul_all(Port,X,Xs):-mul_all('localhost',Port,X,Xs). - (the defaults where our server, which receives the results is running)

mul_all(Host,Port,X,Xs):-all(X,Xs), - executes on remote servers.

remote_run(Host,Port,forall(member(Y,Xs),out(Y))) - executes back on our server.

Host and Port determine the address we want the answers to come back on and the answers are written on the local blackboard from where they can be collected.

Note: in/1 and out/1 are basic Linda blackboard predicates[7]

Places are melted into peer-to-peer network layer forming a 'web of interconnected worlds'. Now different worlds can be grouped into multicast groups. The member of multicast groups viewed as these groups as one world. The concept extends further as groups can intersect and be subsets or supersets and basically have combination of unicast and multicast worlds. The point is that each agents views the world it is interacting with depends on how it is moving its threads to different places.

In the Fig. 1 A, B unicast outputs to two different worlds while C multicasts output to both. The 'unicast all' in E is able to collect from only one blackboard

while 'multicast all' in D collects from both. The **all** operations collect multiple copies of the term **a(3)**, but a sort operation could remove such duplication.

5 Some Properties and Their Consequences

There are various interesting properties of multicast networks, which open up many possibilities for future work especially regarding mobile agents and new architectures for their implementation.

As previously discussed there are three types of mobility in a network software environment:*data mobility, code mobility and computation or thread mobility.* An important shortcoming of computation mobility was that if the thread was providing a service or listening on a particular **(host, port)** it could no longer do so once it moved. In other words, ports are not mobile.

Some properties of multicast addresses and ports overcome exactly this shortcoming. These properties are:

- multicast address and port are same for a multicast group and are independent of host or IP address of the host(*IP Transparent*)
- it is possible to have two servers with same multicast address and port running on the same machine. (*In other words we do not need to investigate if a server with same port and IP is already running.*) Both servers can respond to the request, a client can chose if it wants one or all answers. Also a header can be put, which helps servers discard requests not intended for them.
  This means that when live code or a thread migrates it can just does a *joingroup*[1] on the same group it belonged to and start listening or providing service on the same multicast address and port.

5.1 Impact on Mobile Computers and Transient IP-Address Systems

A mobile computer like a laptop, palmtop etc. does not have a permanent IP-address because one may connect to one's office, home, in an airplane etc. The transient IP address can also come from one connecting through a dialup connection to an ISP. Such systems can launch mobile agents and receive results when connected and hence can be clients [15].

An important impact of the proposed multicast agent architecture on such transient IP-address systems is that they can also *provide a service* and *listen* on a known multicast address and port whenever they are connected to the Internet. This is possible because to listen on a multicast port one's IP address is not needed. One can have any IP address and still listen on the same Multicast address and port.

[1] When we join a group we are telling the kernel, "I am interested in this multicast group. So, deliver (to any process interested in them, not only to me) any datagram that you see in this network interface with this multicast group in its destination field."

Another concept in the Jinni architecture is that of **mobile Linda servants** [28]. A **servant** is an agent, which is launched and on reaching the destination can pull commands from the launching site or other clients, and run them locally.

```
servant:-
  in(todo(Task)),
  call(Task),
  servant.
```

Note that the servant is a background thread and blocks when none of the clients in the group have a task to be done i.e. no 'busy wait' is involved [28].

We will try to expand on these two concepts of multicast and servants to generalize the client/server architecture especially for mobile and transient IP-address systems.

Case 1 Mobile Servers

Even when the mobile server is disconnected it can have **servant agents** running on it, doing the processing for its clients and posting results or queries on the local blackboard. In the mean time, the clients keep making lists of the tasks they want to get done on the server. When the server comes up, the servant can pull the list of tasks to be done by clients and run them. Also the server can have a new IP address but the same multicast address, when the server reconnects. The clients having knowledge of this can collect the required responses from the servers' blackboard.

Case 2 Mobile Clients

Even when disconnected, the mobile client can launch an agent on a different machine, which can do the listening for it. Whenever the client reconnects it can pull list of tasks and results (*the agents do processing*) from the agent and destroy the agent. Whenever the client is disconnecting it can launch the agent again.

This concept can also be extended, as both clients and servers can be made mobile or with unknown or transient IP-addresses with multicasting. As we discussed before, to communicate on a multicast channel we do not need to know the IP. We explore this concept of IP transparent communication further in the next subsection. Some ideas of this mixed mobility of computers and agents are discussed in [6].

This architecture could possibly be three-tier. The applets connect to their server, which offers services to other clients. The server now posts the requests on the local blackboard to be pulled by the applets and executed locally on their machine.

5.2 IP Transparent Architecture for a Platform of Mobile Agents

Consider a group of agents moving freely in the network. Let's assume each agent is a member of two multicast groups: a common shared group address between all agents and a unique personal multicast address. Whenever they propagate they do a *joingroup* on both these multicast groups.

The analogy for private address is that of a cell phone. Each agent can communicate with the others on its private multicast address **being completely unaware about the location of one it is sending messages to.**

The best analogy for the shared common address is that of a broadcast radio channel. Whenever an existing agent spawns a new agent it gives the new agent the set of addresses known to it and the new agent chooses a new private multicast address and communicates to the rest of the agents (via the shared common group address) its private address. Metaphorically, this is like (**broadcasting on the radio channel its cell phone number to communicate with it**)

The main advantage of this architecture is that it carries out communication amongst different agents without any knowledge of each others current location, i.e. no agent requires the knowledge of other's IP address to communicate whether they want the communication to be public within a group or private.

Among the application of a such an architecture could be found in situations where the agents need to communicate with each other but do not have a fixed itinerary or the itinerary changes with the decision the agent makes. The address need not be communicated on each hop but only when a new agent is spawned it needs to make its address known.

For now, the important question of lost messages during transportation of agents remains unanswered. One must not forget multicast is based on UDP and messages can be lost. However, real-time applications like video games, which consider late messages as lost messages could be target applications. Also one of the reliable multicast protocols [2] may be used.

5.3 Fault Tolerant Computing

Here we suggest a protocol, which tries to make sure that even when some agents crash the number of agents is preserved.

Consider the above architecture with each agent having **k** (**k** is a small number greater than 1 say 2 or 3) copies and its own number. Each agent issues a heart beat message on its private channel with its number with a fixed beat time interval (T_{beat}). The agents will have an expiry timeout in case the beat is not received for a particular amount of time from a particular agent (T_{exp}). The agent with the next number in cyclic [**(n+1) mod k**] order generates a new agent with **id** number **n** on expiry of timer (T_{exp}).

Although, each agent is listening for only the previous agent in the cyclic order but even if one agent is left the whole agent group will grow back again. Consider a group of three agents **1**, **2** and **3**. If any two die let's say **2** and **3** then **1** will spawn **3**, which will in turn eventually spawn **2** and hence the group will grow back again.

This makes sure with a good probability that we always have approximately **k** agents on each private channel.

6 Overcoming Agent Location Problem in Absence of Multicast Enabled Routers

In large mobile agent systems agents frequently need to delegate services to other agents. This ability of mobile agents comes from their ability to communicate. This gives arise to the problem of locating a randomly roaming agent for message delivery. In the previous section we proposed an approach based on Multicast which is highly efficient within a LAN but outside is dependent on Network Routers being capable of Multicast.

In this section we propose another approach which is based on Linda Blackboards and does not require any Network Router support.

The problem can be trivially solved in case we assume a home site associated with an agent. In this case, when looking for a particular agent we simply check at the home site where the agent is currently located and deliver the message to it, while the agent updates it's home site every time before leaving for a new site. In this way even if the message arrives before the agent leaves the message is simply posted on the blackboard and when the agent arrives it can check for it's messages.

The real problem arises in large multi-agent systems in which agents have different capabilities and agents receive tasks dynamically and an agent can a receive a task which it can not perform due to incompatible capability or expertise and needs to locate another agent in it's vicinity who can. In our approach as an agent moves from site to site minimally it posts it's presence message on the blackboard (**out(i_am_here(MyID))**). As the agent is about to move it removes it's entry (**in(i_am_here(MyID))**) and posts the next location it is going to (**out(nexthop(MyID, NextAdd))**). Hence, the agent leaves in some sense it's trace. This trace may have mechanism such as timeouts to insure that it is not greater than a definite length. The **ID** of the agent has certain capabilities associated with it. Another agent could do an associative lookup on the blackboard for agents with a particular capability get the **ID** of any agent with a trace on the blackboard and start following the trace to catchup with the agent to delegate a service or deliver a message.

There could be two extremes in the task of locating an agent depending on how the agent performs updates as it moves. The agent could update whole of it's path on a movement, which makes movement an expensive operation. Another extreme is that the agent updates only it's local blackboard, which requires a traversal of it's whole path by every message making message delivery an expensive operation. In our approach we hybridize the two extremes and share update operation task among agents and messages agents and analyze the cost of the update operation, and search for delivery of a message.

In our approach the agent only updates it's local blackboard with it's next hop information The message catches agent does a (**cin(i_am_here(ID))**) which

is an advanced Linda construct translating to an atomic (**rd(i_am_here(ID))-¿in(i_am_here(ID));false**). This makes sure message agent never blocks. In case it fails to find the agent it does an (**in(nexthop(ID, NextAdd))**) and (**out(nexthop(ID, NextAdd))**) and departs to the next hop. If it is able to find the agent it holds the agent as the agent can not move unless it's **ID** is kept back on the blackboard, communicates with the agent, and updates all the sites it had visited till now.

Let's now analyse the complexity of our approach.

(N) maximum distance of agent from current site (maximum possible length of trace of path).
(Tn) Time for movement of agent from one site to another (Processing=large + movement)
(Tm) Time for movement of message from one site to another (Processing =0 + movement)
Assumption: Tn ¿ Tm
(E) Extra Nodes traversed by agent once message starts tracking
Tn * E = Tm * (E + N)
E = N/(Tn - Tm)
Total nodes traversed by the message
= E + N
= N(1 + 1/(Tn - Tm)
Tu Update time of one node Tu = Tm (Processing =0)
Total nodes searched = E + N
Total nodes updated = E + N
Total Time to search and Update = 2 * Tm * N * (1 + 1/(Tn - Tm))

After update operation if the agent has moved K places and a message originating at one of the updated places wants to find the agent then it's
Total Time to search and Update = 2 * Tm * K * (1 + 1/(Tn - Tm)) order of K.

Now if a message comes after agent has moved another K places. then we have two possibilities either it is from one of the places updated in the current step or previous step. In the worst case
Total Time to search and Update = 2 * Tm * (K + 1) * (1 + 1/(Tn - Tm)) order of K + 1

After S such steps the worst case is
Total Time to search and Update = 2 * Tm * (K + S) * (1 + 1/(Tn - Tm)) order of K + S

Now if N is the maximum nodes (Path Length) then we know that K is the nodes it visits in every step and S is the number of steps hence
K*S < N
if we have a good case K = S (approx.) then K+S ¡ 2*sqrt(N)
Time to search and update = 4 * Tm * sqrt(N) * (1 + 1(Tn - Tm))

The worst case messages are sent after **N** moves **(K=N)** of the agent they have to trace all **N** nodes to find the agent (In this case messages are so infrequent that it maybe unimportant). Also, if messages are sent after every move **(K=1)**

and we are sending a message from just updated node the nexthop information is only at the previous node if we have to search for the agent from any node the complexity becomes **N** (but the probabilty of this should become low as **S** increases).

The code for experimentation with the agent location problem appears in the appendix. It also shows how simple it is to experiment with mobile agent algorithms with the Jinni's Mobile Agent Infrastructure.

7 Applications

We will now overview a number of applications likely to benefit from a multicast agent infrastructure.

7.1 Tele-teaching

A set of intelligent agents on student machines, join the multicast group of a teaching server (**run_mul_server**).

The agents can always be given required information or 'todo' tasks from the server as needed on the multicast channel (**remote_mul_run(out(a(X)))**).

The server can collect responses posted on the local blackboards by the agents with the extended blackboard concept (**remote_mul_run(mul_all(a(X)))**) .

The application is more suited to present circumstances as most routers are incapable of multicast. It is however easy to ensure that classrooms are on a single LAN capable of multicast. The main advantage here is that even though the system is interactive the model is not *message based - query/response*. The agents are reactive and intelligent and the responses/queries are posted on the local blackboard from which the server can collect periodically or be informed to collect after a certain time. The model is flexible and can be extended and made more flexible by adding unicast channels and subset multicast groups for teamwork in students.

7.2 Java3D Based Shared Virtual Reality

The three concepts of *intelligent logic programmable agents*, *multicast single step synchronization* and *Java3D visualization* provide an interesting synergy for game programming. We will now explore an implementation architecture we have prototyped on Jinni 2000's multicast layer.

The User's Interface:
The user interface is based on shared Java3D virtual worlds.
Each user can join at anytime by joining a given multicast group.
Each user can create and own objects, which he/she/it can manipulate.
The user is opaque to the knowledge if he/she/it is playing against/with another user or an intelligent agent.

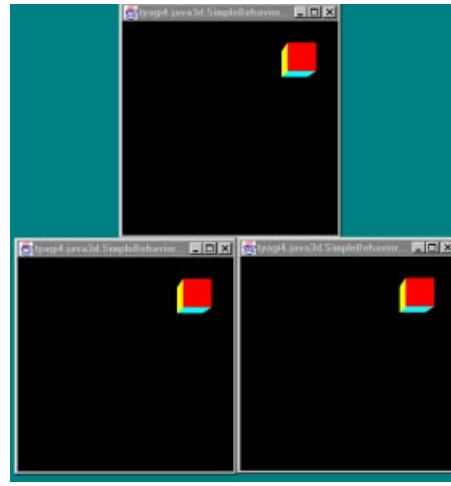


Fig. 2. Synchronized Java3d worlds using multicast.

The Implementation:
The main advantage we have in our implementation is that there is no centralized server. The application is completely distributed. If one user's machine goes down only the objects controlled by him/her go down. This is achieved by having the state being multi-casted to all users and stored only on the local blackboards from where it is to be collected when a new user logs in. The next subsection describes a basic Java3D API on which the virtual world is built and the interface is provided.

The Basic API:
java3d_init initializes and opens Java3d window and joins the multicast group and collects (**remote_mul_run(mul_all(a(X)))**) current state from the blackboards of other users.
new_obj creates a new object and puts the state on local blackboard (**out(a(X))**) and multicasts the predicate call to currently subscribed users.
move_obj moves the object if owned and modifies (**in(a(X)),out(a(Y))**) the state on local blackboard and multicasts the predicate call to currently subscribed users.
remove_obj removes the current object clears entry (**in(a(X))**) from local blackboard and multicasts the change to currently subscribed users.

The blackboards preserve the current state. The multicasting makes sure all updates are single step. The agent scripts are written in Prolog and the visualization is based on Java3D. The logic of agents can be changed and different agents can have different personalities as per the learning logic, algorithm and experience of the agent. The agents generate keyboard and mouse events to play with humans (making them more similar to human interaction).

The Fig. 2 shows three multicast synchronized Java3D worlds, running under our prototype, in three process windows. In a real game they are distributed over the network.

8 Some Problems and Future Work

There are some inherent problems with multicast. The protocol is UDP based (probably because it is not a great idea for each receiver to send an acknowledgment to each sender and flood the network). Also one can never know how many users are currently subscribed to a group. This makes blocking reads (**in(a(X))**) impossible, as we do not know how many responses to loop for. Currently we have implemented multicast **outs**, which do not require responses and non blocking multicast reads (**mul_all**), which collect responses from remote sites and respond on a unicast channel. Some possible scenarios for experimentation would be first matching response or first (k) matching response. Also currently multicast application remains untested on the Internet, as we are confined to the Ethernet LAN. With the new generation routers capable of multicast, it would be interesting to test the applications and protocols over larger domains. The unreliability in the protocol makes it unsuitable for some applications. Our multicast agents work well in real time applications for which a delayed packet is equivalent to a lost packet. Some future work would depend on implementation of reliable multicast protocols and its impact assuming that Internet routers will become more and more multicast aware.

9 Conclusion

We have outlined here an extension of Jinni with a transport layer using multicast sockets. We have also shown some interesting properties of multicast, which have opened various new possibilities for mobile agents and mobile computers. We are currently working on certain applications, which we have shown can be greatly simplified, speeded up and improved with multicast extended version of Jinni. We suppose that the possibilities and applications we have shown here is only a starting point for an unusual niche for Logic Programming based software tools. The spread of multicast technology from simple LANs to the complete Internet and the development of reliable protocols for multicast [2] will necessitate further exploration, to achieve greater insights on mobile agent technology and realize its full potential and possible impact.

Appendix A: Prolog Code for Agent Location Problem

```
%the randomly moving agent
next_step(ID,Ports,Port):-
  %select randomly a port where I wish to go next
  select_port(Ports,Port,SelectPort),
  println(here(ID)),
```

```
  %if I have been here before I need to clean up the old value of next port
  (rd(been_here(ID))->
    in(nextport(ID,_Val))
    ;
    println(firstvisit(ID)),
    out(been_here(ID))
  ),
  out(i_here(ID)),
  %relax do processing
  sleep(10),
  %if i_here(ID) held by some tracking agent then wait for it
  in(i_here(ID)),
  % now leave the port of the next place to be visited
  out(nextport(ID,SelectPort)),
  println(gone(ID)),
  % goto nextport and do the same
  bg(remote_run(localhost,SelectPort,_,next_step(ID,Ports,SelectPort),none,_)).

%selects a randomly generated port
select_port(Ports,Port,SelectPort):-
  remove_item(Port,Ports,RealPorts),
  random(X),
  (X<0->
    X1 is X * -1
    ;
    X1 is X
  ),
  length(RealPorts,N),
  Nex is X1 mod N,
  Next is Nex + 1,
  nth_member(SelectPort,RealPorts,Next).

searchagent(ID,Port,List,Sorted):-
  sort(List,Sorted),
  (cin(i_here(ID))->
    %if agent is here hold it
    println(found(ID)),
    [Currport|Rest]=List,
    sort(Rest,UpdateList),
    %update all the sites traversed till now
    update(ID,Currport,UpdateList),
    %release the agent
    out(i_here(ID)),
    %actually track ends here
    sleep(50)
    ;
    println(searching(ID))
  ),
  println(Sorted),
  in(nextport(ID,Port)),
  out(nextport(ID,Port)),
  println(thenextport(Port)),
  sleep(1).
```

```
%tracks the agent with given ID and builds the list of nodes it has travelled
trackagent(ID,List):-
  searchagent(ID,Port,List,Sorted),
  bg(remote_run(localhost,Port,_,trackagent(ID,[Port|Sorted]),none,_)).

update(ID,Port,Sorted):-
  %update all ports found till now except the current place where we are
  member(P,Sorted),
  not(P=Port),
  println(P),
  bg(remote_run(localhost,P,_,the_update(ID,Port),none,_)),
  fail;true.

the_update(ID,Port):-
  in(nextport(ID,OldPort)),
  println(updated(OldPort,Port)),
  out(nextport(ID,Port)).
```

References

1. A.-R. Adl-Tabatabai, G. Langdale, S. Lucco, and R. Wahbe. Efficient and Language-independent Mobile Programs. In *Proceedings of the ACM SIGPLAN '96 Conference on Programming Language Design and Implementation (PLDI)*, pages 127–136, Philadelphia, Pa., May 1996.
2. K. P. Birman. Reliable Multicast Goes Mainstream. Technical report, Dept. of Computer Science, Cornell University, 1999. Available from http://www.cs.odu.edu/ mukka/tcos/e_bulletin/vol9no2/birman.html.
3. J. Bradshaw, editor. *Software Agents.* AAAI Press/MIT Press, Menlo Park, Cal., 1996.
4. A. Brogi and P. Ciancarini. The Concurrent Language, Shared Prolog. *ACM Trans. Prog. Lang. Syst.*, 13(1):99–123, 1991.
5. L. Cardelli. Mobile Computation. In J. Vitek and C. Tschudin, editors, *Mobile Object Systems - Towards the Programmable Internet*, pages 3–6. Springer-Verlag, LNCS 1228, 1997.
6. L. Cardelli. Abstractions for Mobile Computation. Technical report, Microsoft Research, Apr. 1999.
7. N. Carriero and D. Gelernter. Linda in Context. *CACM*, 32(4):444–458, 1989.
8. S. Castellani and P. Ciancarini. Enhancing Coordination and Modularity Mechanisms for a Language with Objects-as-Multisets. In P. Ciancarini and C. Hankin, editors, *Proc. 1st Int. Conf. on Coordination Models and Languages*, volume 1061 of *LNCS*, pages 89–106, Cesena, Italy, April 1996. Springer.
9. B. Chaib-draa and P. Levesque. Hierarchical Models and Communication in Multi-Agent Environments. In *Proceedings of the Sixth European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-94)*, pages 119–134, Odense, Denmark, Aug. 1994.
10. P. R. Cohen and A. Cheyer. An Open Agent Architecture. In O. Etzioni, editor, *Software Agents — Papers from the 1994 Spring Symposium (Technical Report SS–94–03)*, pages 1–8. AAAIP, Mar. 1994.
11. P. R. Cohen, M. L. Greenberg, D. M. Hart, and A. E. Howe. Trial by Fire: Understanding the Design Requirements for Agents in Complex Environments. *AI Magazine*, 10(3):32–48, 1989.

12. P. R. Cohen and H. J. Levesque. Communicative Actions for Artificial Agents. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 65–72, San Francisco, CA, June 1995.
13. P. R. Cohen and C. R. Perrault. Elements of a Plan Based Theory of Speech Acts. *Cognitive Science*, 3:177–212, 1979.
14. G. Cugola, C. Ghezzi, G. P. Picco, and G. Vigna. A characterization of mobility and state distribution in mobile code languages. In *2nd ECOOP Workshop on Mobile Object Systems*, pages 10–19, Linz, Austria, July 1996.
15. A. K. David Chess, Colin Harrison. Mobile agents: Are they a good idea? Technical report, IBM Research Division, T. J. Watson Research Center, 1995.
16. S. Deering. *Multicast routing in a datagram internetwork*. PhD thesis, Stanford University, Dec. 1991.
17. GeneralMagicInc. Odissey. Technical report, 1997.
available at http://www.genmagic.com/agents.
18. R. S. Gray. Agent tcl: A flexible and secure mobile agent system. In *Proceedings of the Fourth Annual Tcl/Tk Workshop*, pages 9–23, July 1996. http://www.cs.dartmouth.edu/ agent/papers/tcl96.ps.Z.
19. R. S. Gray. Agent tcl: A flexible and secure mobile-agent system. Technical Report PCS-TR98-327, Dartmouth College, Computer Science, Hanover, NH, Jan. 1998. Ph.D. Thesis, June 1997.
20. IBM. Aglets. Technical report, 1999. http://www.trl.ibm.co.jp/aglets.
21. D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko. Agent TCL: Targeting the needs of mobile computers. *IEEE Internet Computing*, 1(4):58–67, July/August 1997.
22. R. Kowalski and J.-S. Kim. A Metalogic Programming Approach to Multi-Agent Knowledge and Belief. In V. Lifschitz, editor, *AI and Mathematical Theory of Computation: Papers in Honour of John McCarthy*. Academic Press, 1991.
23. Y. Lésperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl. Foundations of a Logical Approach to Agent Programming. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI 1037)*, pages 331–346. Springer-Verlag: Heidelberg, Germany, 1996.
24. B. Steensbaard and E. Jul. Object and native code thread mobility among heterogeneous computers. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 68–78, Copper Moutain, Co., Dec. 1995.
25. M. Straer, J. Baumann, and F. Hohl. Mole – a Java based mobile agent system. In *2nd ECOOP Workshop on Mobile Object Systems*, pages 28–35, Linz, Austria, July 1996.
26. P. Tarau. A Logic Programming Based Software Architecture for Reactive Intelligent Mobile Agents. In P. Van Roy and P. Tarau, editors, *Proceedings of DIPLCL'99*, Las Cruces, NM, Nov. 1999.
http://www.binnetcorp.com/wshops/ICLP99DistInetWshop.html.
27. P. Tarau. Intelligent Mobile Agent Programming at the Intersection of Java and Prolog. In *Proceedings of The Fourth International Conference on The Practical Application of Intelligent Agents and Multi-Agents*, pages 109–123, London, U.K., 1999.
28. P. Tarau and V. Dahl. High-Level Networking with Mobile Code and First Order AND-Continuations. *Theory and Practice of Logic Programming*, 1(1), Mar. 2001. Cambridge University Press.
29. N. C. S. Technology and S. Division. Telecommunications:Glossary of TeleCommunication terms:Federal Standard 1037C. Technical report, General Service

Administration Information Technology Sevice, Aug. 1996. Available from http://www.its.bldrdoc.gov/fs-1037/dir-023/_3404.htm.
30. D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *Computer Communication Review*, 26(2), Apr. 1996.
31. D. E. White. A comparison of mobile agent migration mechanisms. Senior Honors Thesis, Dartmouth College, June 1998.
32. J. E. White. Telescript technology: Mobile agents. In Bradshaw [3]. Also available as General Magic White Paper.
33. M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, Department of Computation, UMIST, Manchester, UK, Oct. 1992. (Also available as Technical Report MMU–DOC–94–01, Department of Computing, Manchester Metropolitan University, Chester St., Manchester, UK).

Modern Software Engineering Methods for IP-QoS Resource Pool Management*

Gerald Eichler[1], Falk Fünfstück[2], Fabio Ricciato[3], Anne Thomas[2], Charilaos Tsetsekas[4], and Martin Winter[5]

[1] T-Nova Deutsche Telekom Innovationsgesellschaft mbH, TZ, E24, D-64307 Darmstadt
Gerald.Eichler@t-systems.de
[2] Technische Universität Dresden, Fakultät Informatik, Institut SMT, D-01062 Dresden
{Falk.Fuenfstueck, Anne.Thomas}@inf.tu-dresden.de
[3] CoRiTeL, Via di Tor Vergata, 135, I-00133 Roma
ricciato@coritel.it
[4] National Technical University of Athens, Dept. of Electrical and Computer Engineering, GR-157 73 Athens
htset@telecom.ntua.gr
[5] Siemens AG, ICN WN CC EK A19, D-81359 Munich
Martin.Winter@icn.siemens.de

Abstract. Multimedia applications and IP networks are growing together. This paper proposes to introduce a modular, distributed and scalable Resource Control Layer as intermediary between the application and the network to deliver required quality of service guarantees. The function of the Resource Control Layer is to take care of resource management, resource offering and admission control. It is intended to be an add-on, which does neither affect the underlying network architectures nor the existing applications. Modern software engineering methods are exploited during the analysis, specification, implementation and trial phases. The proposal presented in this paper constitutes a major achievement of an ongoing three-year co-operation project, AQUILA, built by international specialists from providers, manufacturers and research institutes. This leads to practical applicability of the proposed approach to real networks.

1 A Gap between Applications and IP-Networks

Since multimedia is capturing each desk and household, new applications requiring a certain service quality are rapidly developed. Traditionally the applications are themselves responsible to cope with resource bottlenecks known as application level *Quality of Service (QoS)* adaptation. However, future Internet network architectures have to assist resource management in a scalable way at lower level. Well-known QoS aware solutions have already been developed:

– the *Integrated Services (IntServ)* model, which makes reservations using the Resource Reservation Protocol (RSVP),

* This work is partially funded by the European Union under contract number IST-1999-10077 "AQUILA".

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp 19–30, 2001.

– the *Differentiated Services (DiffServ)* model, which supports relative prioritisation of IP traffic.

This paper adopts a DiffServ-oriented approach in a lightweight manner.

Today, there is still a gap between networks, already able to differentiate traffic according to the DiffServ approach, and applications with demand for QoS. What is missing, is a scalable entity, which forms network services out of DiffServ *per hop behaviours (PHB)* or *per domain behaviours (PDB)*. Furthermore, this entity should have means to control the network elements in order to assign the requested service to the customer. In order to fill such a gap, the presented proposal introduces a **Resource Control Layer (RCL)**, logically distributed in the network elements at network management level as well as in the end-user system at application level (fig. 1). This supports the QoS requirements of existing applications without modifying them.

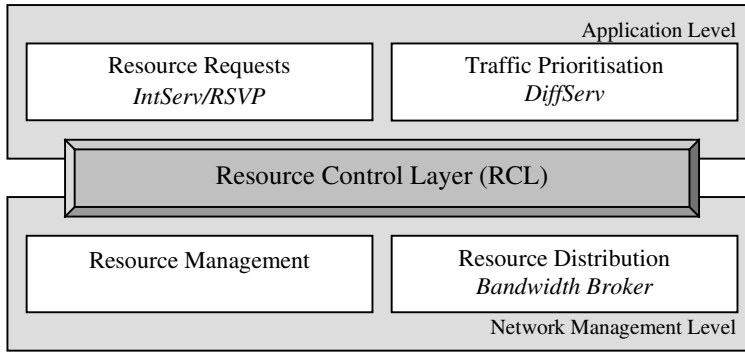


Fig. 1. Resource Control Layer. As intermediary between application and network the RCL does not touch existing entities. It forms a distributed overlay architecture.

The high speed of the progress of Internet technologies makes it necessary to develop the RCL under permanent observation of the current trends. Therefore, the RCL software is developed in two main phases, and the results are going to be evaluated during two trials. On the one hand, this makes it possible to get early prototypes in order to verify the RCL architecture. On the other hand, rapid changes of the Internet's scientific and market environment can be considered for the next prototypes.

In detail, the software engineering process follows, as depicted in fig. 2, a *spiral model* like [4] with two lifecycles, where general work products are the specification of requirements for the RCL architecture, the design of the RCL components, their implementation and integration, and the test results of the two trials. Each work product is documented in a deliverable.

For the software development itself, the widely accepted *Unified Modelling Language (UML)* is used to support a distributed software development process [11]. This is done in combination with *Computer Aided Software Engineering (CASE)* tools to ensure coherency between model, source code and documentation.

An international project team was formed to take the challenge of efficient resource management in distributed systems: the AQUILA project [3]. Scientists from network

and content providers, manufacturers, universities and research institutes came together to design and implement a resource control layer architecture by means of modern software technology methods. This paper presents selected project achievements.

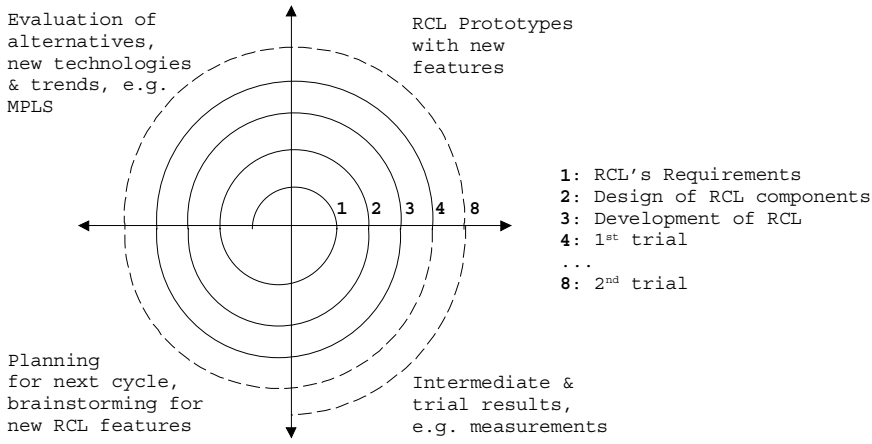


Fig. 2. Software engineering process represented as a spiral model. Requirement analysis, design specification, development and implementation as well as trial phases follow the same sub-steps [4].

The organisation of the paper is as follows: Chapters 2 to 4 are divided in two main sections each: one describing the technical focus and the other describing the used methodology and software technology, respectively. High-level system specification and implementation support for IP systems are taken into consideration. First, the RCL model is introduced. Second, it is discussed how QoS is offered to the customer. Third, the mechanisms used to distribute available network resources are presented. Background information about the AQUILA project, chapter 5, and proposals for advanced features, chapter 6, complete this paper.

2 Resource Control Layer Model

2.1 Resource Control Layer Architecture

The RCL performs authentication of users and authorisation of requests. It keeps track of the resource usage within the network and performs admission control based on this information. The combination of resource and admission control is an important prerequisite to avoid traffic congestion.

The RCL consists of three types of entities as depicted in fig. 3: the Resource Control Agent, the Admission Control Agent and the End-User Application Toolkit.

Per domain, one **Resource Control Agent (RCA)** keeps track of the resource usage within the network and distributes the resources to the ACAs. This distribution is done on a rough level to allow the ACAs to act autonomously within defined limits.

The **Admission Control Agent (ACA)** performs authentication, authorisation and admission control. The ACA takes care of local resources. Each edge router has its own associated ACA instance. ACAs communicate among each other as well as to the above RCA.

The **End-User Application Toolkit (EAT)** offers the network services to the customer, and translates and forwards QoS requests from the application to the ACA. Different mechanisms to request QoS like DiffServ, host marking or RSVP are supported. Modular *proxies* are provided for special applications, which are not QoS aware natively. All accepted requests are mapped to appropriate network services as described in chapter 3.

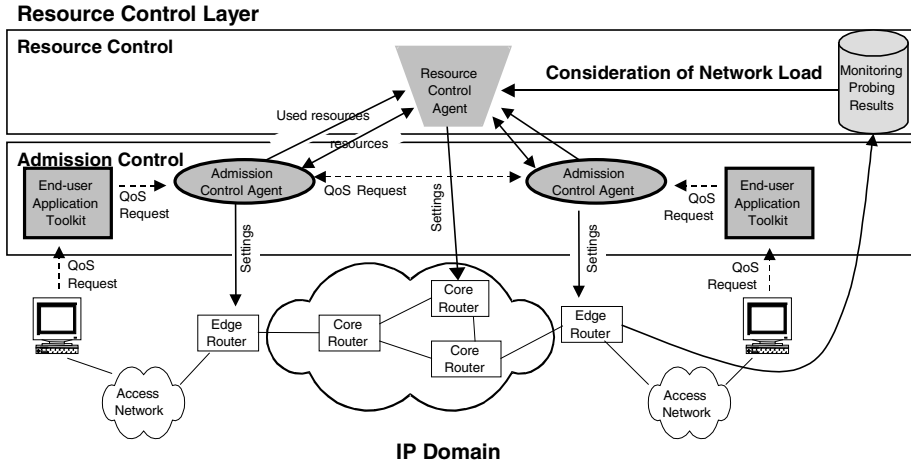


Fig. 3. Layered resource control architecture. QoS requests are sent to the Admission Control Agent by means of the End-user Application Toolkit. The Resource Control Agent keeps track of the resources used in the network, taking into account both, the user requests and monitoring of the actual resource utilisation. The original network architecture and elements remain unchanged.

RCA, ACA and EAT build a distributed system [16]. The design of this system follows three main goals:

1. Locality: Tasks are located close to where they are required: the EAT near the hosts and the ACA near the network edge. The RCA, shown as a single entity in fig. 3, may itself be distributed and control the network resources in a hierarchically structured way.
2. Robustness: All elements are designed, so that failure of one element only degrades or does not affect at all the operation of the other entities or the performance of the system.
3. Security: Security relevant functions are separated from other functions, so that sensitive information can be clearly hidden from the end user.

Altogether, the elements of the resource control layer act as a kind of distributed bandwidth broker [9], which offers its services in a user-friendly and versatile approach to the customer. Besides the static distribution of resources, measurement tools use probes to analyse the current network situation concerning QoS parameters,

e.g. loss, delay and delay variation with high precision. The RCL uses a hardware abstraction layer to support a wide range of currently used routers and existing network infrastructure.

2.2 Resource Control Layer Development

The architectural approach, namely the distribution into three RCL entities led to a modular software architecture, i.e. each of the three entities is realised as a separate "component" with:

- an exactly defined subtask of the RCL functionality (as mentioned above),
- well-defined interfaces to communicate with other components, and
- the integration of a mechanism that ensures an fault tolerant, independent work of the component.

The interfaces are specified programming language independently with the *Interface Definition Language (IDL)*, and the communication between the different components relies on the *Common Object Broker Architecture (CORBA)*, the OMG's vendor-independent architecture and infrastructure that computer applications use to work together over networks [10]. The ACA, for example, provides a "resource request" interface towards the EAT including parameters that allow the specification of QoS and traffic values to be requested for an application. In fact, this is the only way to request QoS from the RCL.

The implementation of the RCL components is made in the platform independent programming language *Java*. The project uses further modern technologies such as the *Lightweight Directory Access Protocol (LDAP)* for the retrieval of network configuration data from a central server [15], the *eXtensible Markup Language* (*XML)* for the specification of application relevant data [14]*, Java Server Pages (JSP)* for the realisation of user-friendly GUIs [2].

The following chapters are focused on two important mechanisms used by the RCL, namely the resource offering (chapter 3) and the resource management (chapter 4). It is also described how software engineering can support their development. Well-known software *analysis and design patterns* [7, 8] have been recognised and utilised in the original version or slightly adapted to the specific case.

3 Resource Offering

3.1 Mapping between Application Requirements and Network Resources

The RCL core function is to dynamically assign network resources to requesting applications. The network offers a limited number of transport options for user IP traffic: the **Network Services (NS)**. The NS are meant to be defined by the network operator and offered to its customers in the context of *Service Level Specifications (SLS)*.

For the AQUILA architecture the following six NSs have been pre-designed to meet the diverse requirements of a wide range of applications:

- Premium Variable Bit Rate (PVBR) => jitter sensitive, variable flows, e.g. mpeg4,
- Premium Constant Bit Rate (PCBR) => jitter sensitive, constant flows, e.g. VoIP,
- Premium Multimedia (PMM) => streaming applications, premium data transfer,
- Premium Mission Critical (PMC) => loss-sensitive applications,
- Custom NS => request for special parameters, not covered by the previous NS,
- Best Effort (BE) => no guarantees, e.g. e-mail.

Each NS provides a specific combination of QoS performance parameters (e.g. delay, loss) suitable to a particular kind of data (e.g. streaming multimedia, mission critical etc.). Also, each NS exactly defines the set of traffic parameters (like peak rate, token bucket, etc.) that must be passed in the QoS request, and the allowed range of values for them. Given the complexity of the parameter set, the formulation of a QoS request to the network would be a challenging task for the user, even for an expert one. Moreover, users have a different notion of quality, which is based mainly on perception and not on technical parameters. They are mainly concerned about issues such as the quality and the size of the video image or the clearness of the voice. Evidently, some simpler and more comprehensible means are needed to allow users to place their QoS requests. This is accomplished by the EAT. Two levels of abstraction have been defined in the EAT: the application level and the network level. The translations between the two level parameters are done by the **converter module** inside the EAT.

At the **application level**, the end-user chooses QoS with a verbal description corresponding to a universal apprehension of applications. Considering the sample case of a video-conferencing session, the user can formulate its QoS requests by selecting from a set of proposals the size of the picture, the frame rate, the sound quality etc. These parameters relevant to the application level are referred to as **session characteristics**.

At the **network level**, a QoS request represents a list of network specific parameters related to low-level traffic and QoS parameters such as peak rate and delay variation. All those parameters are referred to as **technical characteristics**, and are included in the **QoS request** message which is sent by the EAT to the ACA, along with the specification of the selected NS.

The converter module performs a two-way mapping between the application level and network level parameters. In this effort, the converter module makes use of the concept of the **application profile**. An application profile contains all the relevant session and technical characteristics of an application, as well as rules for the translation between them.

An example scenario of the operation of the converter module is the following: When the user wishes to place a QoS request for a specific application, the converter module, based on the profile of the application, calculates all the possible sets of session characteristics. Those calculations take into account parameters such as: the kind of application and its service components, the Service Level Agreement set between the user and the provider, the configuration of the user's host, etc. After the different options have been presented to the user and a selection has been made, the converter module maps the session characteristics into a QoS request message to be sent to the ACA.

3.2 Data Structure for Application Profile

The mapping tasks between application level and network level are based on sets of parameters corresponding to the application profiles. Therefore, it was necessary to define a structure describing the application in order to be able to store the parameter sets.

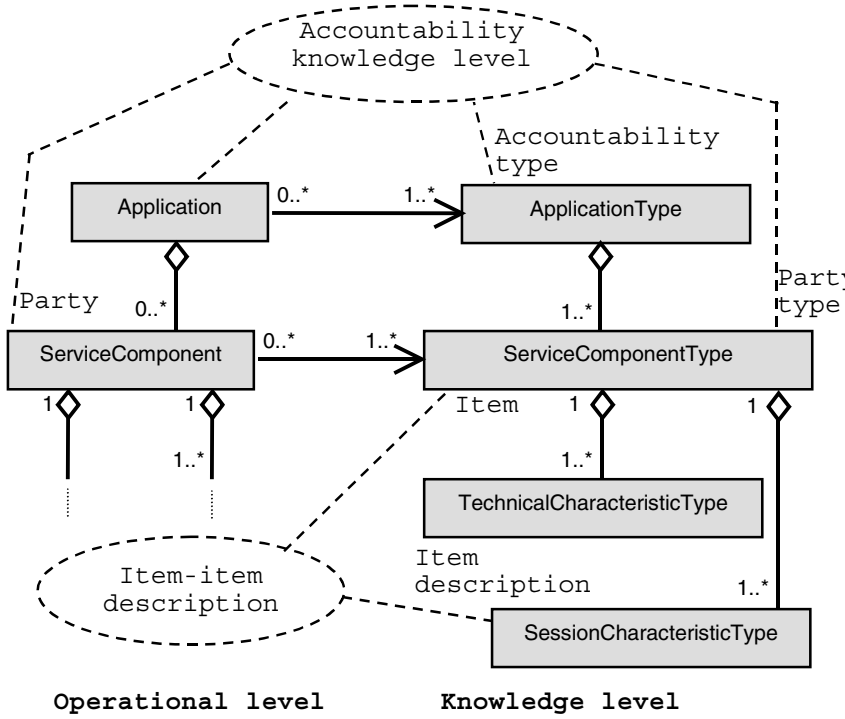


Fig. 4. Application profile class diagram. The adapted item-item description analysis pattern and the adapted accountability knowledge level analysis pattern are depicted using UML [11].

Applications can be described from a general point of view and from a concrete one. First, at a general level, an Application can be associated to ApplicationTypes like "video conferencing", "voice over IP", "streaming media", etc. The "video conferencing" ApplicationType for example is described by the ServiceComponentTypes: "video", "voice" and "data". A ServiceComponentType itself is described on the one hand by network-oriented TechnicalCharacteristicTypes, e.g. the codecs, on the other hand by the user-oriented SessionCharacteristicTypes like "picture size", "sound quality", etc. The slightly modified item-item description analysis pattern [1], see fig. 4, allows the modelling of such relations between objects and shared descriptions where an item object is described by a description item. In this case: a ServiceComponentType (the description item) describes an Appli-

cationType (the item object). SessionCharacteristicTypes and Technical-CharacteristicTypes describe a ServiceComponentType, etc.

Second, the analysis of concrete applications leads to the following statement. A concrete application like NetMeeting by Microsoft shares with similar applications the same "video conferencing" ApplicationType. As previously seen, such an ApplicationType can be modelled at a general level. The structure describing the concrete application (e.g. NetMeeting) should follow the same description rules as the one of the ApplicationType. The adapted accountability knowledge level analysis pattern [7], see fig. 4, allows the split of the overall model in two sections: an operational one (corresponding to the concrete application), and a knowledge one (corresponding to the application type). At the knowledge level the model records the general rules that govern the structure and correspond to a meta-level description of an application profile.

4 Resource Management

4.1 Resource Distribution and Algorithms

It was shown in the previous chapter that any application QoS request is mapped into a QoS request for the ACA by the converter module. In the following it is assumed that any QoS request corresponds to an amount of bandwidth for a specific NS. The mapping is done partially at the EAT (by the converter module which chooses the relevant NS) and partially in the ACA (which computes the "effective bandwidth" B from the declared traffic parameters). Thus, the only relevant resource for each NS is *bandwidth*.

In order to get sufficient scalability, the ACA must be able to process the QoS requests locally. To accomplish that, an NS-based *provisioning* scheme is adopted: for each NS j each **Edge Router (ER)** ER i is assigned a fixed amount of bandwidth $l_i^{(j)}$, called AC Limit. The ACA responsible for ER i can accept incoming QoS requests for NS j as long as the sum of the bandwidth demands for the accepted flows remains below the limit $l_i^{(j)}$. The value of the AC Limit as well as the bandwidth demand for each active reservation is logically stored at the ACA. The initial computation of the set $\{l_i^{(j)}\}$ is a matter of global optimisation taking into account the expected traffic distribution.

In order to adapt to fluctuations in offered traffic, it would be desirable to have some dynamical sharing of resources (i.e. bandwidth) between ERs and/or between NSs. The AQUILA architecture introduces the **Resource Pools (RP)** to allow for some degree of dynamical sharing of resources between ERs. Inside an RP and for any NS the ERs can exchange the assigned bandwidth between each other. The identification of RP sets depends on the network topology and is related to the presence of bottlenecks. Application of the RP concept is straightforward in the case where a set of ERs is connected in a star to a **Core Router (CR)**, as depicted in fig. 5(a). The value of 18 Mbps between node 4 (the RP "root") and the core represents the bottleneck, which limits the total traffic that can be injected by ERs 1 to 3 (the RP "leaves"). The RP root maintains a counter indicating the amount of *available* resources, defined as the difference between the maximum injectable traffic and the

total bandwidth currently assigned to its RP leaves. For each RP leaf $a_i^{(j)}$ denotes the "used" resource, i.e. the sum of the bandwidth demands of all the currently "accepted" reservations. For each ER i and for each NS j an upper watermark $u_i^{(j)}$ and a lower watermark $d_i^{(j)}$ ($u_i^{(j)} > d_i^{(j)}$) will be defined, as well as the following "behaviour rules" for the leaves:

1. *Request Rule: whenever the amount of used resources $a_i^{(j)}$ raises above the upper watermark ($a_i^{(j)} \geq u_i^{(j)}$), ask the root for a block of more resources.*
2. *Release Rule: whenever the amount of used resources $a_i^{(j)}$ falls below the lower watermark ($a_i^{(j)} \leq d_i^{(j)}$), return a block of resources to the root .*

Of course, sharing of resources is subject to some additional constraints, for example those relevant to the link capacity of the links between the root and the leaves. The choice of the amount of resources to be requested or released (the "block" sizes) as well as the setting of the watermarks are details of the algorithm. Note that any resource request/release represents a transaction between a leaf and the root of the RP, thus a message overhead. In general by tuning parameters such as block size and watermarks one can tune the frequency of such transactions, in order to achieve some good compromise in the trade-off between message overhead and resource utilisation.

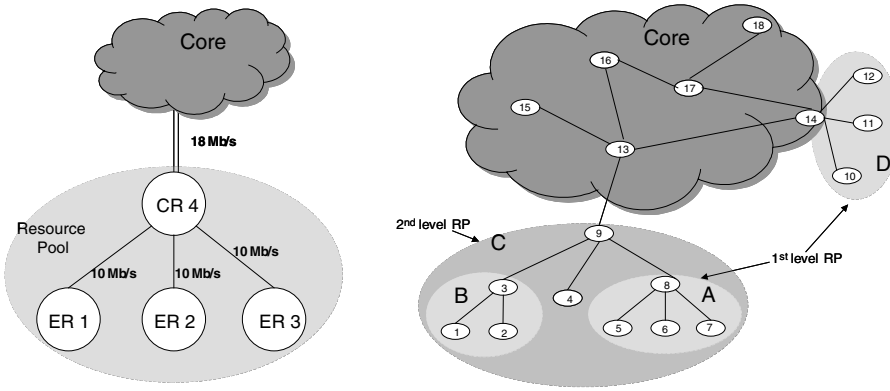


Fig. 5. Resource Pools. (a) Example: The indicated rates represent the maximum bandwidth allocatable to a generic NS *j* for each link. (b) Example: Resource Pools hierarchically organised.

The above approach can be hierarchically extended to build RPs, whose elements are not ERs but other RPs. This case is depicted in fig. 5(b), where RP "C" is composed by node 9 as root and RPs "A" and "B" as leaves. The hierarchical RP approach can be straightforwardly applied in those networks, whose external areas represent a hierarchically structured topology, which is a quite common case in real networks.

4.2 Hierarchy Concept of Resource Pools

The data structure underlying the resource distribution algorithm, namely the RP, is an important concept of the presented solution. One activity of the software process consists in the recognition of well-known analysis and design patterns, beside the transcription of the requirements and concepts for the RCL in UML diagrams (activity diagrams, sequence diagrams and class diagrams) for a better common understanding.

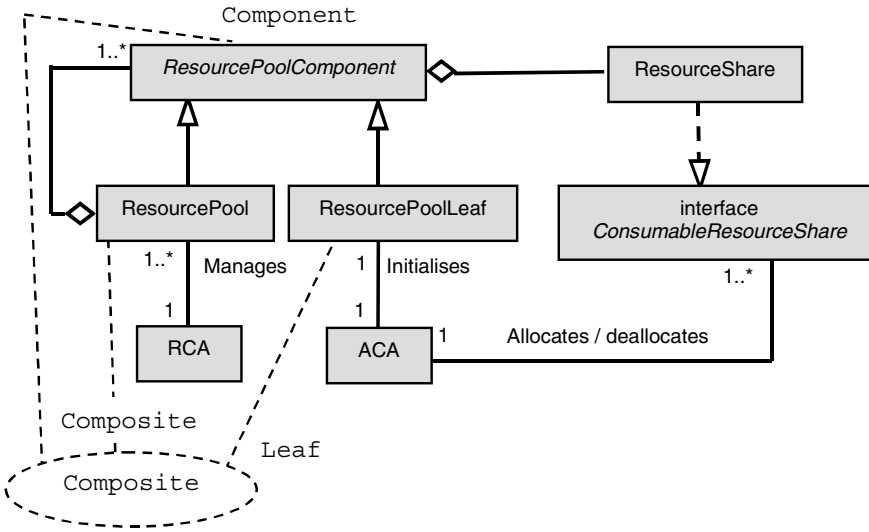


Fig. 6. Resource pool UML class diagram. It shows the composite pattern, the Resource Pool as composite, the resource pool element as component and the ACA as leaf.

In the case of the RP, the composite structure design pattern [8] can be applied to represent it as tree structures to indicate part-whole hierarchies.

The composite pattern is indicated in the UML class diagram fig. 6 representing the RP model. An RP is hierarchically organised as a tree: a ResourcePool is composed of ResourcePoolComponents that can either be another ResourcePool or, at the deepest level, ResourcePoolLeafs.

5 The AQUILA Project

The AQUILA (Adaptive Resource Control for QoS using an IP-based Layered Architecture) project [3] is a European research project partially funded by the IST programme [6], placed in the "Next Generation Networks" cluster. Well-structured, three workpackage groups, each consisting of several workpackages, bundle the activities of requirement analysis, specification and simulation, design and implementation as well as integration, trial and exploitation following a defined

milestone plan. Close co-operation with the IST projects CADENUS [5] and TEQUILA [13], which cover other QoS aspects, is performed.

The AQUILA project supports an interdisciplinary research and implementation teamwork testing modern software engineering methods. The partners are: Siemens (D), Q-Systems (GR) as manufacturers; Telekom Austria (A), Telekomunikacja Polska (PL), Elisa Communications (FIN), T-Nova Deutsche Telekom (D) from the ISP and operator side; Bertelsmann mediaSystems (D) as content provider, and from the universities and research institutes: Dresden University of Technology (D), National Technical University of Athens (GR), Salzburg Research (A), Warsaw University of Technology (PL) and CoRiTeL (I).

The AQUILA project lasts until end of 2002. It will have two trial phases following the spiral approach to verify the RCL results.

6 Outlook

6.1 Multi-domains and Other Advanced Features

The initial approach follows a single-ISP solution. One RCA keeps track of the network resources per domain. For multi-ISP scenarios an inter-communication of RCAs of different domains will be introduced.

Further introduction of some form of dynamical resource sharing between NS ("vertical" sharing) is envisioned as a further extension to the model. Thus, an RP identifies a set of ERs that can be donors to each other. In the simplest formulation, a RP is a "closed" set, in the sense that its elements ERs cannot exchange resources with other ERs outside the RP. More sophisticated forms of "non-closed" RPs could represent further extensions of the method.

The default settings and parameter ranges of pre-defined NS will be optimised during two trial phases. An Internet Draft on SLS [12] was sent to the IETF. The definition and implementation of a common platform-independent API for QoS request is envisaged.

Measurement driven control loops will support the distribution of resources with a certain QoS guarantee. This influences the dynamic re-configuration of ERs. The concept is open to add additional RCL entities, e.g. to control CRs, while already designed components remain untouched.

6.2 Project Conclusions

Although the AQUILA project is composed of specialists of very different disciplines and interests it has been noticed that the partners have actively and successfully used the modern software engineering techniques, e.g. the adapted spiral model, UML, CORBA & IDL, Java to develop the RCL components of their responsibility, to communicate among each other, and to exchange the results.

The first implementation and integration tests are ongoing, and the first integration results of components of the RCL has shown that systematic use of such technologies increases the effectiveness, in comparison to similar former project experiences with

distributed and international co-operation. Therefore, the best pre-conditions are met for the first trial phase.

References

1. Ambler, S. W.: Building object applications that work. SIGS books, Cambridge University Press (1998)
2. Avedal, K. (ed.): Professional JSP. Wrox Press Ltd. (2000)
3. AQUILA consortium: Adaptive Resource Control for QoS Using an IP-based Layered Architecture. AQUILA homepage, URL: http://www-st.inf.tu-dresden.de/aquila/
4. Boehm, B.: A Spiral Model of Software Development and Enhancement. IEEE Computer, vol. 21, #5, May 1988, pp 61-72 (1988)
5. CADENUS consortium: Creation and Deployment of End-User Services in Premium IP Networks. CADENUS homepage, URL: http://www.cadenus.org/
6. European Commission: Information Society Technologies Programme (IST). IST homepage, URL: http://www.cordis.lu/ist/
7. Fowler, M.: Analysis Patterns. Addison Wesley (1998)
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Reusable Objected-Oriented Software. Addison-Wesley Publishing Company, Reading Massachusetts (1995)
9. Neilson, R. (ed.): A Discussion of Bandwidth Broker Requirements for Internet2 Qbone Deployment. Version 0.7 (work in progress),. URL: http://www.merit.edu/internet/working.groups/i2-qbone-bb/doc/BB_Req7.pdf
10. Object Management Group (OMG): CORBA 2.4.2 specification (formal/2001-02-01). URL: http://www.omg.org/technology/documents/formal/corbaiiop.htm
11. Object Management Group (OMG): UML Resource Page. URL: http://www.omg.org/technology/uml/index.htm
12. Salsano, S. (ed.): IETF draft on Definition and usage of SLSs in the AQUILA consortium. URL: http://www.ietf.org/internet-drafts/draft-salsano-aquila-sls-00.txt, November 2000
13. TEQUILA consortium: Traffic Engineering for Quality of Service in the Internet, at Large Scale. TEQUILA homepage, URL: http://www.ist-tequila.org/
14. W3C Architecture Domain: Extensible Markup Language (XML). URL: http://www.w3.org/XML/
15. Wahl, M., Howes, T., Kille, S.: Lightweight Directory Access Protocol (v3), IETF rfc 2251, URL: http://www.ietf.org/rfc/rfc2251.txt, December1997
16. Winter, M. (ed.): System architecture and specification for the first trial. AQUILA deliverable D1201, Munich (2000)

Adaptive Quality of Service Management Using QoS Proxy and User Feedback for Wireless Links

Dang-Hai Hoang*, Dietrich Reschke, and Werner Horn

Technical University of Ilmenau,
D-98693 Ilmenau, Germany
{Hoang,Reschke,Horn}@prakinf.tu-ilmenau.de

Abstract. In this paper, we propose a mechanism for Quality of Service (QoS) Control with a proxy model for wireless communication networks. The concept is characterized by the management of user profiles in the proxy server, and the adaptive adjustment of the quality of service with respect to user's requirements presented by the user profiles and the user feedback. The QoS proxy server resides on the base station of a wireless network and intermediates the user's mobile terminal and the other side such as another mobile user or an Internet server in the wired network side. According to the available bandwidth on the wireless link, the proxy server adjusts QoS before sending data stream to the mobile user based on his user profile. The user also can express his preference during the connection by sending an user feedback to the proxy server to get more or less quality. On the other hand, the proxy server has to manage the total bandwidth and its distribution, furthermore to realize QoS adaptation according to the actual link bandwidth condition

1 Introduction

By the initiation of mobile multimedia services, including the mobile internet access, high technical orders are required on the underlying communication systems. The End-to-End Quality of Service (QoS) is in the center of the availability of services. Until now, most of the works have been in the developments of individual network components. Summaries are given in [1][2]. A key observation is that the QoS guarantee is not solely dependent on the capability of the end-systems, but also on the capability of networks and inter-systems together. Therefore, these two factors and the interaction between them should be considered in an unified concept.

Due to the varying transmission characteristics of the wireless environments, the providing of QoS guarantee in wireless networks is rather limited. QoS of wireless networks can vary over time in contrast to wired environments.

From the network' point of view, the QoS guarantee is the ability of the networks to fulfill the different request of the applications of customers. For this reason, two

* with a George-Forster Research Fellowship of Alexander-von-Humboldt Foundation.

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp 31–40, 2001.

possibilities can occur. If the networks cannot satisfy the request of the users due to resources lacking, the applications are to adapt to this situation. As shown in literature, an amount of QoS aware applications have been developed which have the capability of adapting to their environments. The other possibility is the QoS on-demand approach [3], in which the user can actively react to achieve as a better QoS as possible. In this case, there is a need for controlling the user feedback and for controlling actual QoS characteristics of the networks.

In this paper, we propose a mechanism for QoS Control with a proxy model. In this model, the user preference in form of an user profile is sent and stored in a QoS Management Server (also called Proxy Server) connected to the base station of the wireless networks. The concept is furthermore characterized by the dynamic monitoring of the available link bandwidth and the adaptive adjustment of QoS in the actual link condition. One key characteristic of the concept is in the QoS control by the mobile user itself. The mobile user can send his user feedback for expressing his preference on QoS to the proxy server, which then adjusts the QoS as required by the user based on the actual link condition.

The paper is organized as follow. In section 2, we present the concept framework of QoS control using proxy structure and the user feedback interface. Section 3 shows the components of the concept. In section 4, we discuss the method of QoS monitoring by bandwidth information from receiver and by calculating the packet loss probability based on gathering the queue lengths. Section 5 presents the measurement for a video transmission, in order to demonstrate the benefit of the concept. Finally is the conclusion of the paper.

2 The QoS Management Framework

2.1 Overview

At first, an entire picture over QoS Structures in heterogeneous networks is represented. Two areas can be regarded roughly: wired and wireless area. QoS is an end-to-end matter. From the view of the network, there are three important parameters: delay, delay jitter and packet loss. The well-known QoS technologies [5][6] are IntServ, DiffServ, MPLS (Multi-Protocol Label Switching) and BB (Bandwidth Broker).

For wired area, the reasons for QoS degradation may be congestion in the network, lacking resources, unsuitable scheduling. In case of congestion, the packets may be lost and retransmission is needed [7]. Another possible solutions for congestion avoidance are traffic engineering, load balancing [8]. In order to provide QoS, the required resources should be reserved and allocated by the network. A re-negotiation is possible during the session. Other solution is the choose of suitable scheduling and buffer/bandwidth management [9]. In addition, different concepts were proposed to support QoS in end-systems (see [2] for a summary). These concepts have been proposed for process management, resource management in end-system (e.g. CPU, Buffer, I/O Interfaces).

In contrast to the main reason of the losses in wired networks, the packet losses in wireless networks occur not only caused by the congestion, but also due to possible high BER (Bit Error Rate) of the radio channels, fading and interference between channels etc. In this case, the radio links can have instable channel characteristics and temporal bandwidth variation. The usual retransmission for recovering lost packets is not suitable and not efficient any longer. The Forward Error Correction (FEC) has disadvantage of complex overheads. New solutions are necessary. To date, there are improvements through radio link technologies, new and efficient compression techniques, etc. Some of the works have proposed in improvements of the protocol stacks, such as protocol booster [10]. The other improvements are in end-systems themselves, i.e. the applications themselves. These applications are called QoS aware applications as indicated for instance in [11].

Our approach to QoS control uses a proxy structure which is in contrast to the other works that the proxy server adaptively adjusts the QoS according to the actual link bandwidth condition and the user preference / user feedback. We use the notation QoS management server which means the same as the notation proxy server or shorter QoS server.

2.2. Description of the Proposed Concept

The simple concept model is shown in figure 1 and 2. The proxy server is connected to the base station of a wireless network. The receiver is a mobile terminal such as a handheld computer, the sender may be an another mobile terminal or a host in the wired network.

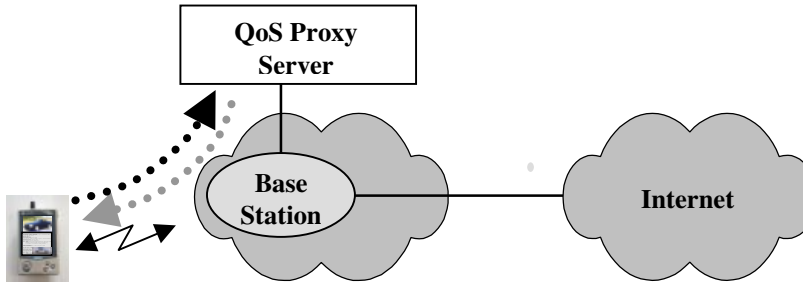


Fig. 1. Modeling of the concept

We demonstrate the concept using a video transmission from the sender to the receiver. The processing call is as follows. The sender wants to setup a call to the receiver, thus it sends a connection setup message to the call management of the wireless network with the address of the receiver. We suppose that the call signaling was done by the wireless network and a connection is setting up. Now, the receiver has to send a short message containing his preference and his user profile is stored respectively in the proxy server. The information stored in the user profile are codec, frame rate, display size, color depth, etc. Using this information, the proxy server set a queue for the connection in the application layer of the base station and set up a correspond-

ing weight for the connection with respect to other active connections. The sender application admits a video stream to the respective queue in the base station. This data stream is then transcoded according to the user profile, i.e. the required QoS of the receiver and is sent to the receiver terminal. At the base station, the proxy server marks the data packets with respect to important and unimportant video frames. During the connection, the proxy server monitors the available link bandwidth and discards the unimportant frames in case of lack of bandwidth. If the link bandwidth is very scarce, or for example when the receiver is behind a building and is not possible to reached, it is typically for the proxy server to discard all video frames, thus the receiver can only receive the audio tone. In other case, the user wants to get better quality of service as possible or to receive the quality he wants to pay for. To realize this possibility, a new user interface is to develop which enables the user to express his preference by clicking on a button on the display window. Upon his clicking, a user feedback signal is sent to the proxy server. The proxy server can vary the weight of his connection queue, thus he can receive more bandwidth in compare to other connections, and change the frame rate of his connection.

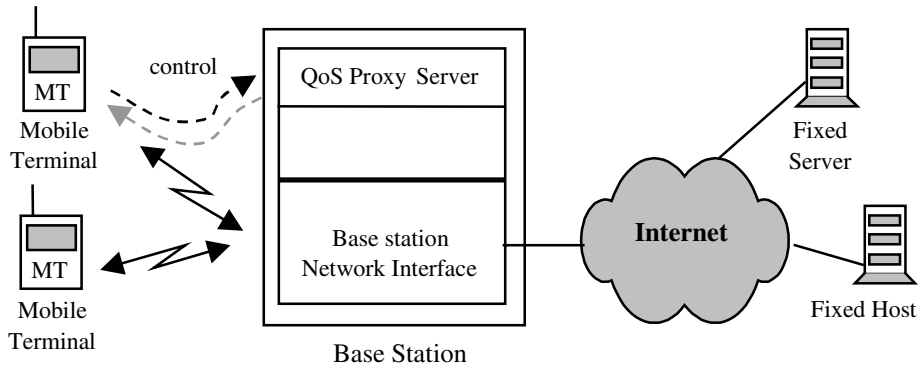


Fig. 2. QoS management architecture

3 Components of the Concept

The concept of QoS Management consists of the following components: the sender / receiver application including. the user interface, the queue system, the user profile and the QoS proxy server. Figure 3 shows the demonstration of the concept for a video transmission.

3.1 User Interface

The simple user interface is depicted in figure 4 and 5. The graphical interface mainly consists of a video screen area and three buttons. The call button allows user to make a

call, by clicking on this button, the user wants to enter the desired destination address of the far end customer, either the IP address or the name of computer to be called to. By clicking on this button again, the call is disconnected. The other button is designed for setting user preference. The user can select his preference before any call occurs and the user preference is transmitted to the base station upon the requirement of the proxy server. It is also possible for the user to send the preference during the call. By clicking on the Quality button (minus or plus, i.e. worse or better) the user can send his feedback expressing the desired quality to the proxy server.

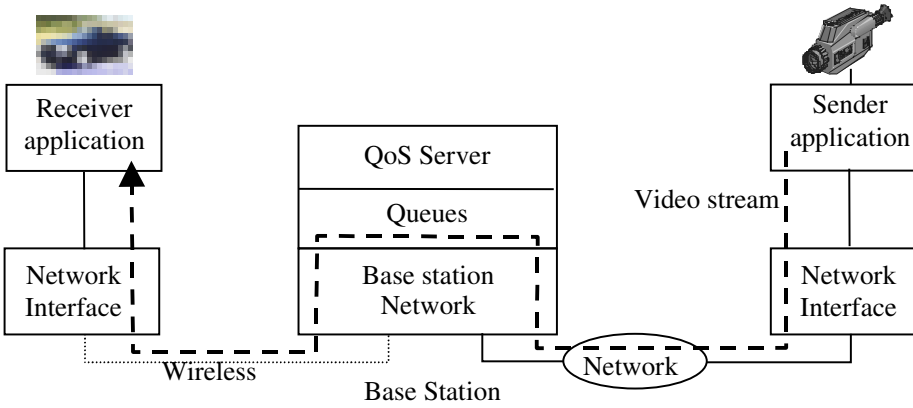


Fig. 3. An example for the demonstration of the concept

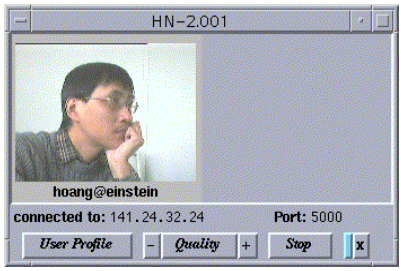


Fig. 4. The Graphical User Interface

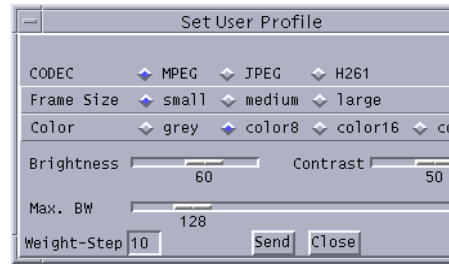


Fig. 5. Window for setup User Profile

Sender application: The sender application allows the user to establish a call by clicking on the call button and following enter a destination address. The sender application then captures a video stream from the equipped video camera and sends this video stream to the base station, in the corresponding queue set up by the proxy server for this connection.

Receiver application: The receiver application enables the user to set up the preference before any call occurs. This user preference is sent to the proxy server and will be stored in an user profile in the proxy server. During the call, the receiver application allows the user to activate a feedback signal to the proxy server by clicking on the

quality button. Upon receiving the transcoded video stream from the base station, there is a task to decode the data stream and to play out the video stream. One key task is the measure of link bandwidth and periodically sending this to the proxy server.

3.2 The Queue System

The queue system resides in the proxy server. There is a separate queue for each down link connection in the system. We consider a reference time slot system according to 16 time slots of a W-CDMA (Wideband Code Division Multiple Access) frame. In the packet mode, the base station has to select a packet from a connection during each down link time slot. By giving different weights on queues, we can determine the order of packet from different connections on the transport layer of the base station. Upon user feedback, the weight of the desired connection can be changed. The queue system has the task to select the packets from queues according to their weights, thus a virtual time slot ordering can be realized. The incoming packets are marked according to important or unimportant frames, in order for the discarding of unimportant frames in case of lack of bandwidth.

3.3 The User Profile

The user profile contains information about the preference of user's mobile terminal. This information expresses the technical characteristic of the mobile terminal equipment such as the display size, the color depth, etc. Furthermore, it includes the receiving preference of the customer, for example the desired frame rate, codec, image size, maximum bandwidth. The figure 5 shows a typical window for setting the user preference. Beside of the mentioned parameters, the user can also choose the weight step which is the step whenever the user clicks on the minus or plus button. Actually, the weight can be adjusted from the default value 2 (see figure 6) to 100, that is the percentage the user can get in according to the maximum link bandwidth. The default value for weight step is currently 10 and can be modified by the user.

3.4 The QoS Proxy Server

The proxy server has above all the task to receive and store the user preference in form of an user profile. According to this information, it set up the weight for the customer's connection and the user preference. It receives the encoded video stream from the sender, transcoded this video stream according to the format desired by the receiver depending on the user preference, and sends the transcoded video stream to the receiver. Figure 6 presents the control window of the proxy server for three video connections.

The proxy server receives periodically the information about the link bandwidth measured by the receiver and adjusts the quality of connection to the actual link condition. For example, if the link bandwidth becomes scarce, it will decrease the frame

rate, discard unimportant frames, decrease the color depth, or even discard all video frames, thus only voice is transmitted in this worse case. The information about actual bandwidth is sent from the receiver to the proxy server via the base station using a special packet with the packet header BANDWIDTH_INFO.

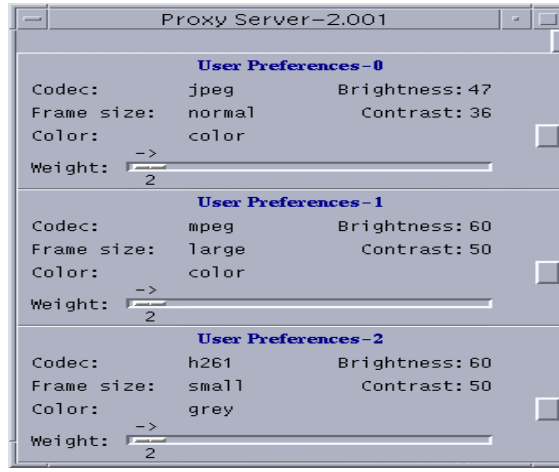


Fig. 6. The control window of the proxy server for 3 connections

By clicking on the quality button, the user sends a special message with the header USER_FEEDBACK to the proxy server. This expresses the desired quality of service of the customer. The general way is to send only a parameter indicating the desired quality, for example a utility value. The proxy server has then to map this value to the network parameter such as bandwidth, delay, loss etc. The mapping should realize a translation. A simple way to do, which we suggest current is that, the user will send the feedback packet with an explicit wish on quality, e.g. frame rate, image size, etc. The weight for the user connection will be adjusted depend on the positive or negative feedback, i.e. clicking on plus or minus QoS - button. We are investing on the development of a quality function to generally express the quality desired by the user. This quality function should express the utility of the user including the bandwidth, the efficiency of the transmission protocol, the error rate of link. This is the topic of next works.

4 QoS Monitoring

Figure 7 shows the principle model for monitoring the QoS in the proxy server. As shown in the figure, the proxy server manages a queue system, each queue for one connection. Incoming packets from different flows arrive in the queue system. Each queue is reserved for a flow. Scheduling of the packets occurs on the application level. By giving different weights on queues, we can determine the serving order of packets from different flows, thus the quality of connection can be controlled. Without this

scheduling the packets will be served in a First-In-First-Out (FIFO) manner. Moreover, the queue lengths are gathered and sent to the QoS proxy server. Depending on the packet loss probabilities calculated by the queue lengths and the information feedback about the occupied bandwidth from the receiver, the proxy server can monitor the QoS from different packet flows, i.e. the quality perceived by users. With both types of information, the server can make a decision respect to QoS guarantee in the actual link condition, and the required QoS from the user via user feedback.

The scheduling algorithm was defined in [12] as follows:

$$ST_i = \max(TA_i^k, FT_{i\_})$$

$$FT_{i+} = ST_i + \frac{L_k}{g_i(t)}$$

where *ST* is the start tag, *FT* the finishing tag of the queue *i*, *TA* is the arrival time that is the time whenever a packet *k* becomes the head of a queue. L_k is the packet length of the packet *k*, *g* is the weight of queue, *i* is the index of queue.

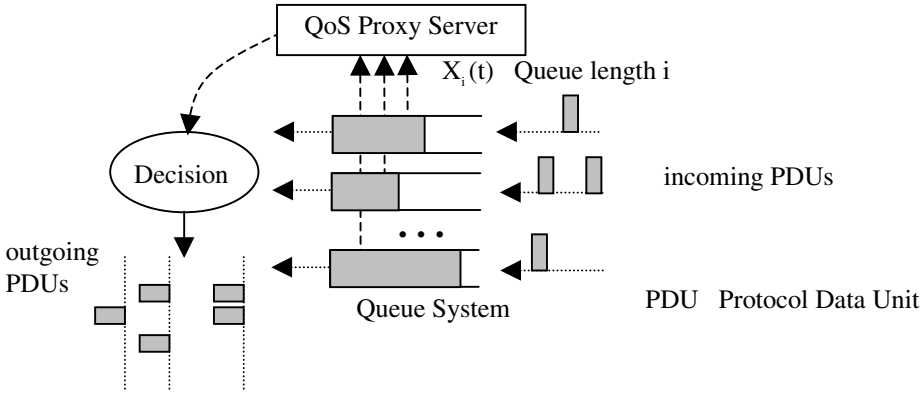


Fig. 7. The concept model for queue system and scheduling at the application level

The relation between the bandwidth allocation and the queue length can be described by the following equation: *C(t) = B * (1 + X(t))* [12], where B is a constant, X is the queue length of a queue. The probability of packet loss P is equal $P(X = X_m)$, where X_m is maximal queue length of a queue. That is the probability, by which the maximum queue length is exceeded.

5 Experiment Results

For demonstration of the concept, we have experimented using a laboratory testbed consisting of three computers: a video sender, a video receiver and the proxy server. Each computer is a Sun machine connected to each other over the Local Area Network. Even through there is not really a wireless network environment, it is possible to

show the concept with the setting of the user preference and receiving and adjusting QoS based on the user feedback. Three video connections were set up, in order to compare the frame rate of the connections and to demonstrate the weight control. The result is shown in Fig. 8. For the connection 1, the weight was increased by step of 10 (it is the default value and it means 10% of the maximum bandwidth the user can receive). at times t=40s, t=60s, t=100s and decreased at time t=140s, t=170s according to the clicks on the plus or minus QoS-button, respectively. The adjustment step is in fact an implementation issue and is calculated depends on the available bandwidth. The other connections have constant weights with the initial value of 2.

As shown in the figure, the frame rate, i.e. the QoS of the connection 1 is better by increasing weight, whereas the frame rate of other connections 1 and 2 remain stay. In fact, there is an oscillation interval before the frame rate becomes stable during each change of the video image. The duration of this vibration interval is depending on the video compression methods.

At each change of the weight, the frame rate is shortly low due to the updating in the system.

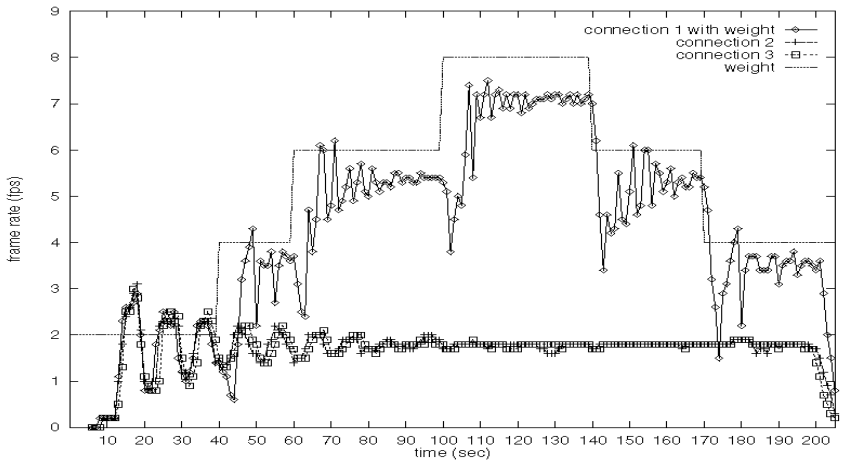


Fig. 8. Frame rate according to the setting weight for the connection 1 in compare to the connection 2 and 3

6 Conclusion

In this paper, we presented the difficulties in providing QoS over the wireless multimedia communication networks and showed various proposed concepts for overcoming the problems.

Due to the QoS on-demand approach, we propose a QoS control concept using proxy structure and user feedback in wireless multimedia communication. In sec-

tion 2, we presented the architecture of the concept supporting video transmission over wireless link, and described in section 3 the components of the concept. We discuss in section 4 the possibility with QoS monitoring using bandwidth information sent back by receiver and by calculating the packet loss probability based on queue lengths gathered by the proxy server. We also present the concept for flow scheduling on the application level at the proxy server for QoS control. In section 5, we demonstrated the concept for video transmission by an experiment with three video connections. We showed the result of adjusting weight depending on the user feedback, thus a higher quality, i.e. an increased frame rate, can be received. We are investigating further on the design of the control mechanism and on the development of a quality function to realize the translation from user feedback to network QoS.

References

[1] CCH94, A.Campbell, G.Coulson, D.Hutchison, "*A Quality of Service Architecture*", ACM Comp. Commun. Rev. Apr. 1994.

[2] ACH98, C.Aurrecoechea, A.T.Campbell, L.Hauw, "*A Survey of QoS Architectures*", Multimedia Systems, 6.98, 1998.

[3] ETSI TR 122 960, V3.0.1(2000-01: *"Universal Mobile Telecommunications System (UMTS); Mobile Multimedia Services including Mobile Intranet and Internet Services"*, (3G TR 22.960 version 3.0.1 Release 1999).

[4] ETSI TS 123 107, V3.2.0 (2000-03*): "Universal Mobile Telecommunications System (UMTS); QoS Concept and Architecture"* (3G TS 23.107 version 3.2.0 Release 1999).

[5] QoSForum.com, Stardust.com, White Paper, *"QoS Protocols and Architectures*", 8.99, qosprot_v3.doc, 1999.

[6] QoSForum.com, Stardust.com, White Paper, *"Introduction to QoS Policies*", 6.99, qospol_v11.doc

[7] JA99 R. Jain, *"Quality of Service in Data Networks, CIS788-99*", 1999.

[8] XN00, X.Xiao, L.M.Ni, *"Internet QoS: A Big Picture", 2000.*

[9] KUR96, J.Kurose*, "Open Issues and Challenges in Providing Quality of Service Guarantees in High-Speed Networks"*, 1996.

[10] FMSB98, DC:Feldmeier, AJ.McAuley, JM.Smith,et.al., *"Protocol Boosters"*, IEEE Journal on Sel. Areas in Comm. Vol.16,No.3, Apr. 1998.

[11] BRS00, M.Bechler, H.Ritter, J.H.Schiller, *"Quality of Service in Mobile and Wireless Networks: The Need for Proactive and Adaptive Applications"*, IEEE Hawaii Intl. Conf. On Syst. Sciences, HICSS00, 2000.

[12] HO00, D-H.Hoang, D.Reschke*, "An Adaptive Priority Approach to Bandwidth Scheduling in High Speed Networks to the Support of Multimedia Applications"*, IFMIP 2000, June 2000, Hawaii.

Sharing Extendible Arrays in a Distributed Environment

Tatsuo Tsuji, Hidetatsu Kawahara, Teruhisa Hochin, and Ken Higuchi

Department of Information Science, Fukui University, 3-9-1 Bunkyo, Fukui-Shi, 910-8507, Japan

Abstract. We describe a distributed system for scientific applications based on sharable extendible arrays. In our sharing scheme, a part of or all of an extendible array placed on the server side can be shared among many clients. A sharing array can be declared local to a client, and it initially shares an arbitrary part of an extendible array on the server side. After the initial sharing, it can be extended locally on the client. The locally extended part on the client side together with the shared part on the server side can be logically handled as a single extendible array. A set of extendible arrays can be persistently stored on secondary storage. In this paper, after proposing the properties of extendible arrays and their manipulation, we propose the notion of sharing an extendible array. Then the language constructs and the runtime routines for managing extendible arrays and sharing arrays are described stressing their scope and visibility in the client/server system architecture.

1 Introduction

Scientific applications often require computation on multi-dimensional arrays for modeling and analyzing scientific phenomena. The strong need to handle large scale scientific data efficiently have been promoting extensive research themes on organization or implementation schemes for multi-dimensional arrays on computer memory or secondary storage [1,2].

Along with these researches, in recent years, on line analytical processing (OLAP) employing multi-dimensional arrays is becoming increasingly important as today's organizations frequently make business decisions based on statistical analysis of their enterprise data. Since these data are ordinary multi-dimensional and requires analysis along each dimension, multi-dimensional arrays also play an important role on such OLAP applications. A fundamental demand of OLAP systems is extremely fast response times for multi-dimensional queries issued against these multi-dimensional arrays [3,4].

In addition to the element access speed, operational capability on multi-dimensional arrays with its language aspect is another important demand of such scientific applications or OLAP applications. For example, [5] proposes array manipulation language and describes user-defined operations on multi-dimensional arrays. [6] gives formal model of sub-array (tile) based operations on multi-dimensional arrays and suggests their effective implementation. In most

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 41–52, 2001.

of these researches however, the employed multi-dimensional arrays are of fixed size in order not to sacrifice the accessing speed obtained by fast addressing functions.

On the contrary, an *extendible array* can expand its size dynamically in any directions during execution time [7,8]. While a *dynamic array* [9] is newly allocated entirely when required at the execution time, all the existing elements of an extendible array are used as they are; only the extended part is dynamically allocated. In general, an extendible array is extendible in any direction without any relocation of the data already stored. Such advantage makes it possible for an extendible array to be applied into wide application area including the above mentioned where necessary array size cannot be predicted and can be varied according to the dynamic environment during execution or operating time of the system.

The difficulty to implement an extendible array is that the ease of extension along an arbitrary direction is not compatible with high memory utilization or with construction of an addressing function that enables fast random accessing to an array element. The *index array model* described in [10] is superior in element access speed gained at the cost of only small memory space for the index array. The superiority over its competitors such as [7] and [8] is demonstrated in element accessing speed and memory utilization.

In this paper, based on the *index array model* [10,11,12], we propose a scheme of sharing extendible array, and describes a distributed system architecture employing the scheme. As far as we know there are no previous works discussing sharing of extendibe arrays, which is important for DBMS, OLAP or other systems based on extendible arrays. In our sharing scheme, a part of or all of an extendible array placed on the server side can be shared among many clients. A *sharing array* can be declared locally to a client, and it initially shares arbitrary part of an extendible array on the server side. After the initial sharing, it can be extended locally to the client. The locally extended part on the client side together with the shared part on the server side can be logically handled as a single extendible array. The initial sharing and the extensions afterwards can be performed by each client if necessary.

Firstly, the index array model for realizing extendible arrays is briefly explained. Secondly, after proposing some properties of extendible arrays by further elaborating the index array model, we propose the notion of sharing an extendible array. Thirdly, the language constructs and the runtime routines for managing extendible arrays and sharing arrays are described stressing their scope and visibility in the client/server system architecture. The last section is devoted to the conclusion.

2 Model of Extendible Arrays

An n-dimensional extendible array A has two kinds of auxiliary table on each extendible dimension $i (i = 1, \ldots, n)$. See Fig. 1. One is the *history table* H_i that memorizes extension history and the other is the *address table* L_i that holds the

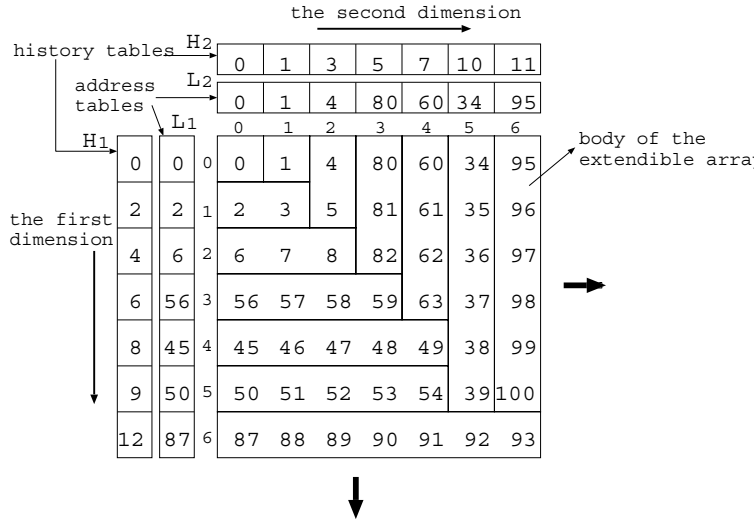


Fig. 1. Realization of a two dimensional extendible array

first address of each *sub-array* allocated dynamically at extension. In addition to these auxiliary tables, A has the *history counter* h that stores the current history value. If A extends its size by one along the i-th dimension, an $(n-1)$-dimensional fixed size sub-array is allocated and h is incremented by one. Then the value of h is stored on H_i and the first address of the sub-array is stored on L_i. Since h monotonously increases, H_i is an ordered set of history values. In Fig. 1, h is 12.

Using these two kinds of table the address of an array element can be computed as follows. Consider the element $[3,4]$ in Fig. 1; we assume that the index of the elements starts at 0. Compare $H_1[3] = 6$ and $H_2[4] = 7$. Since $H_1[3] < H_2[4]$, it can be proved that the element $[3,4]$ is involved in the extended sub-array v occupying the address from 60 to 63. The first address of v is known to be 60, which is stored in $L_2[4]$. Since the offset of $[3,4]$ from the first address of v is 3, the address of $[3,4]$ is determined as 63. Note that we can get such a simple computational scheme to access an extendible array element only by preparing small auxiliary tables. The superiority of this scheme is shown in [10] compared with other schemes such as hashing [8].

3 Sharing Scheme

In this section, we propose some important properties of an n-dimensional extendible array based on the index array model, and define the *sharing* of an extendible array.

Property 1 Let the current *shape* of an n-dimensional extendible array A be $[D_1, D_2, ..., D_n]$ and let the value of the current history counter be h. For

a past history value $p \in I_h = \{0, 1, ..., h\}$, A's shape $[d_1, d_2, ..., d_n](0 \leq d_i \leq D_i)$ at that time can be determined uniquely. Namely, the function $f : I_h \longrightarrow \prod_{i=1}^{n}\{0, ..., D_i\}$ exists. Here $\prod$ denotes the Cartesian product of sets.

For each i of $1 \leq i \leq n$, let p_i be the max history value in H_i not exceeding p. Each d_i can be determined as the value such that $H_i[d_i] = p_i$.

Example 1 Let $p = 7$. In Fig. 1, the max values in H_1 and H_2 not exceeding p are 6 and 7 respectively. Since $H_1[3] = 6$ and $H_2[4] = 7$, $d_1 = 3$ and $d_2 = 4$. Hence the A's shape at the history value p is $[3, 4]$, so $f(p) = [3, 4]$.

It should be noted that the above property is important since the shape of any n-dimensional extendible array (i.e., a vector value) can be uniquely restored by only knowing the corresponding past history value (i.e., a scalar value). A typical application of this property will be found in the later program example in Section 5.

Definition 1 Let the current history counter value of an n-dimensional extendible array A be h, and let p be $0 \leq p \leq h$. Let f be the function specified in Property 1. We denote the extendible array $f(p)$ as A_p and the relation between A_p and A as $A_p \prec A$.

[Table Copy]
Let A be a two-dimensional extendible array whose element size is s. Assume that the array B is the portion of A from l_1 to u_1 in the first dimension and from l_2 to u_2 in the second dimension. See Fig. 2. In order to make B work as an extendible array, it is necessary to prepare the history table and the address table for each dimension of B. Let the history tables of A be H_1^A and H_2^A, and the address tables of A be L_1^A and L_2^A. Let the history tables of B be H_1^B and H_2^B, and the address tables of B be L_1^B and L_2^B.

H_1^B and H_2^B can be obtained by copying the corresponding portion of H_1^A and H_2^A. Since the order of extensions can be preserved in the B's history tables thus copied, the tables also work well at the address computation of B's elements.

Now consider the case of accessing the element $B[i, j](0 \leq i \leq u_1 - l_1, 0 \leq j \leq u_2 - l_2)$ in the B's *coordinate*. Suppose $H_1^B[i] \geq H_2^B[j]$, so $B[i, j]$ is involved in the sub-array starting at $L_1^B[i]$. Then the address of $B[i, j]$ can be computed as $L_1^B[i] + j \times s$. Also suppose that $A[i', j']$ is the element corresponding to $B[i, j]$ in the A's coordinate; i.e., $A[i', j']$ and $B[i, j]$ are identical and their addresses are the same. The address of $A[i', j']$ can be computed as $L_1^A[i'] + j' \times s$. Hence $L_1^B[i] = L_1^A[i'] + (j' - j) \times s$. Since $j' - j = l_2$, the offset that should be added to $L_1^A[i']$ at copying the address table becomes $l_2 \times s$ (in the case where $H_1^B[i] \leq H_2^B[j]$, the offset becomes $l_1 \times s$). Note that the above discussion on the table copy can be easily generalized on an n-dimensional extendible array.

Definition 2 Let the current value of the history counter of an n-dimensional extendible array A be h, and for p of $0 \leq p \leq h$, let $A_p = [d_1, d_2, ..., d_n]$. Let all the partial arrays of A_p be denoted by $partial(A_p)$, namely

$$partial(A_p) = \{\prod_{i=1}^{n}\{l_i, ..., u_i\} | 0 \leq l_i \leq u_i \leq d_i\}.$$

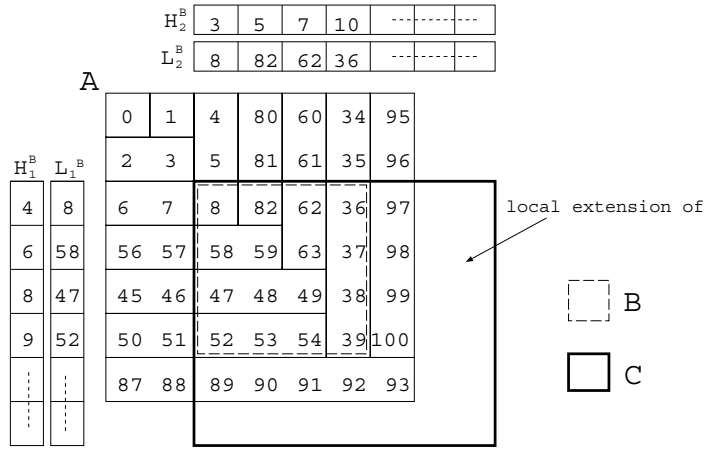


Fig. 2. Sharing array defined on the extendible array in Fig. 1

From the above definition, $\{f(i)|1 \leq i \leq p\} \subseteq partial(A_p)$ immediately follows.

Property 2 Let the current value of the history counter of an n-dimensional extendible array A be h. For an arbitrary partial array B in $partial(A_h)$, an extendible array C can be constructed such that $B \prec C$.

C is specifically called as a *sharing array* on A with the *initial share* B. A is called as the *host array* of C.

Example 2 Let A be a two dimensional extendible array shown in Fig. 1, and let B be a partial array of A. A sharing array C on A with the initial share B is shown in Fig. 2.

The storage for C's two kinds of table is allocated separately from those of A. The extended part of C after its initialization, namely the storage occupied by $C - B$, is also allocated separately from that of A.

A sharing array can be declared locally to a client, and it initially shares arbitrary part of an extendible array on the server side. After the initial sharing, it can be extended locally to the client. Note that the locally extended part on the client side together with the shared part on server side can be logically handled as a single extendible array. The history tables and the address tables of a sharing array are usually placed on the client side, and the address calculation of a sharing array element is done on the client side, so that the work load of the server would be reduced. See Fig. 3.

Example 3 Fig. 4 is a graph database represented by three adjacent matrices G, $G1$ and $G2$. G is stored by using an extendible array on the server side storage. $G1, G2$ are stored by using two extendible sharing arrays on the clients Client1 and Client2 respectively. Initially, G is shared by $G1$ and $G2$. Afterward Client1

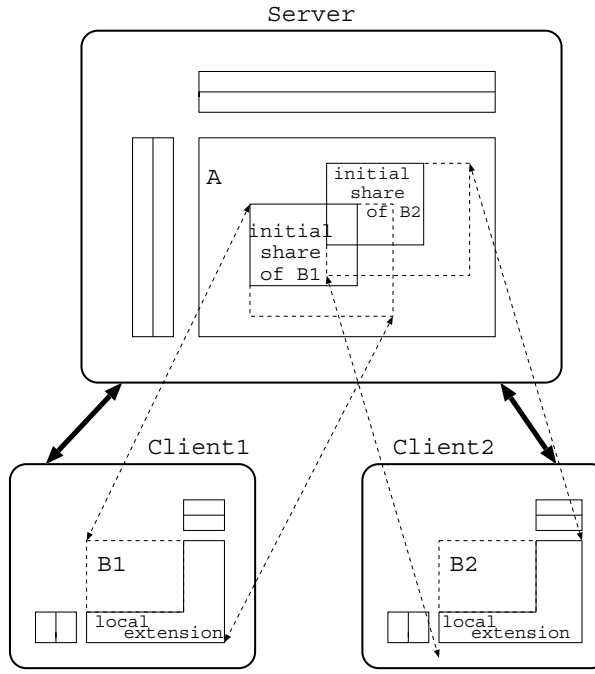


Fig. 3. The system structure

and Client2 locally add the nodes $\{f, g\}$ and $\{f, g, h\}$ and the related links to $G1$ and $G2$ respectively.

Note that Client1 cannot observe the subgraph on Client2, and vice versa since the extensions are allocated using their own local storages. On the contrary the modifications of the shared subgraph on Server can be effected on $G1$ and $G2$, and observed by both Client1 and Client2. The extension on each client may be shrunk after working on it. Note also that G can extend its size using the server side storage independently of the extensions of $G1$ and $G2$.

Reducing and Resizing

While an extendible array is extendible in arbitrary dimension, it is also reducible in arbitrary dimension. Few literatures treat the shrinking of extendible arrays.

The shrinking of an extendible array is performed in one of the two ways; *reducing and resizing*. The reducing retores its past shape specified by its history value p of $0 \leq p \leq h$. The reduction is done by going down the history table of each dimension to check the history values and deallocates the storage of each sub-array whose history value is greater than p. For example, in Fig. 1, let p be 7. Since $h = 12$, the sub-arrays of the first dimension whose history values 12,9,8 are deallocated, and the sub-arrays 11,10 of the second dimension are deallocated.

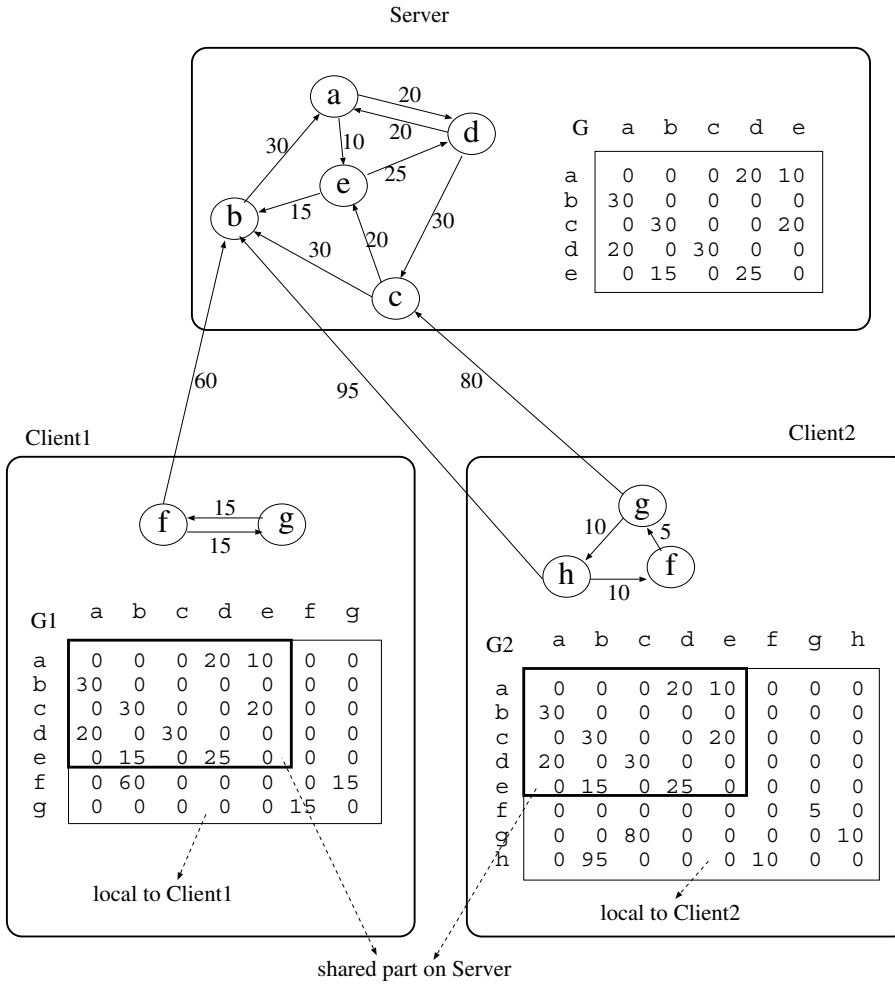


Fig. 4. A graph database

The other way of shrinking, namely resizing, is fully general. The shrinking is not restricted to one of the past shapes, but can shrink to any one of the partial arrays specified by $\{\prod_{i=1}^{n}\{0, ..., u_i\} | 0 \leq u_i \leq d_i\}$. Moreover the resizing can extend or shrink to arbitrary size on every dimension. In the resizing, the size of every dimension after the resizing should be specified.

Fig. 5 shows an example of the resizing. The portions of the storage (a) and (b) are deallocated by the resizing, but note that the actual storage deallocation of (c) is deferred until the resized array has reduced its size not to contain any elements of the sub-array involving (c). (d) and (e) are newly allocated for extension after shrinking along the first dimension.

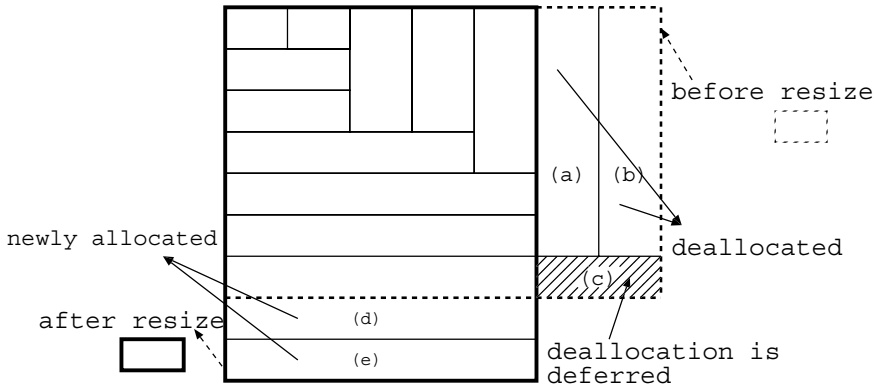


Fig. 5. Resizing of an extendible array

4 Language Constructs and Scope of Extendible Arrays

In this section, we introduce new constructs to the C language that define extendible arrays and sharing arrays. Scope and visibility of extendible arrays in the distributed environment are also described. A program written by using these constructs are converted into the corresponding C program invoking the runtime routines explained in the next section.

– Creation of an extendible array

scope storage_mode element_type extendible_array_name[*initial_size*:] ... [*initial_size*:] *permission*;

‘`:`’ denotes that the corresponding dimension is extendible and the void of ‘`:`’ denotes that its size is statically fixed. Hence the traditional fix sized arrays can be defined by avoiding ‘`:`’ in every dimension. *scope* specifies the scope of extendible arrays. If *scope* is ‘`public`’, the extendible array is placed on the server side. It can be accessed and shared by every client process if the process is compatible with the specified *permission*. The absence of ‘`public`’ means that the extendible array is placed on the client side. If an extendible array is declared globally, it can be accessed and shared by every function in the client process.
If *storage_mode* is ‘`persistent`’, the extendible array is allocated on secondary storage. Otherwise, it is allocated on memory.
The creator of an extendible array is its owner. *permission* is specified like in the UNIX file permission mode. The designators `r`, `w` mean that the extendible array is readable and writable respectively, and `x` means that it is extendible and reducible.

– Reference of an extendible array

An extendible array can be referenced by prefixing '`extern`' on the name of an extendible array. The following construct declares the reference of the *extendible_array_name* array stored on the server side.

`extern public` *extendible_array_name*[ ] ... [ ];

If the '`public`' is absent, the *extendible_array_name* array placed on the local client side is referenced.

– Definition of a sharing array

storage_mode *element_type* *sharing_array_name*[:] ... [:];

sharing_array_name is the name of a sharing array which shares a part of an extendible host array on the server side or the local client side. Note that the initial *shape* of a sharing array is not specified at this declaration. The binding of a shared array to a host array will be done at runtime using the runtime routine `share()`, in which the initial shape is specified. In `share()`, the element type compatibility and the validity of the initial shape will be checked by comparing with those of the host array. Note also that *storage_mode* is effective only on the locally extended storage part on the client side.

5 Manipulation of Extendible Arrays and Sharing Arrays

Extendible arrays or sharing arrays can be manipulated using the client side or the server side runtime routines. We explain only the major ones.

– `extend()`, `reduce()`
`extend()` is the function that extends an extendible array or a sharing array.

`extend`(*array_name*, *extension_mode*, *extension_size*, ..., *extension_size*);

extension_mode determines whether the *extension_size* is specified absolutely or relatively to the current size. The current value of the history counter after extension is returned.
`reduce()` is the function that reduces an extendible array or a sharing array.

`reduce`(*array_name*, *history_value_before_extension*);

The second parameter denotes one of the past history values. Note that Property 1 in Section 3 guarantees that the shape of an extendible array at some point of past time can be uniquely restored. `reduce()` returns the value of the history counter after reducing.

– `bound()`
`bound()` is the function that tells the current size of the specified dimension of an extendible array or a sharing array.

`bound`(*array_name*, *specification_of_dimension_or_0*);

If the second parameter is 0, `bound()` returns the current value of the history counter of the extendible array or the sharing array.

Example 4 The following is a typical example program that utilizes the combination of the above runtime routines.

```
public persistent char ea[10:][10:] (rwx rw r);
   /* create a persistent extendible array on the server side */
main()
{  int mark, u1, u2;

   ......................
   mark = bound(ea, 0);     (1) /* memorize the current shape */
   extend(ea, ABS, u1, u2); (2) /* extend absolutely */
   ......................
   /* working by using the extension on the client side */
   ......................
   reduce(ea, mark);        (3) /* reduce to the size before
                                   extension */
}
```

(1) The current history value counter of extendible array `ea` is memorized in order to restore its *shape* before extension.
(2) `ea` is extended absolutely to the size `[u1,u2]` and the extension is used together with the part before extension.
(3) The extension is removed after the work.

– `share()`,`unshare()`
Sharing a part of an existing extendible array with *sharing array* is specified at run time using `share()`. To read or write the initial share of the host array, the client process must have the corresponding permission.

`share`($array\_name1, array\_name2, reduce\_mode, lower\_bound, upper\_bound,$
$..., lower\_bound, upper\_bound$)

The sharing array $array\_name1$ would be bound to the corresponding part of the storage for the host array $array\_name2$. $reduce\_mode$ specifies the behavior when a portion of the host array shared by the sharing array is reduced by another client in the multi-client environment. Note that the element positions of a sharing array can be specified by using its own coordinate not by the one of its host array.
`unshare()` cancels the sharing. The storage of the extension allocated on the client side after `share()` would be automatically deallocated. Moreover, locking on the storage of the corresponding part of the host array would be automatically unlocked.

– `lock()`
A client process must acquire a *lock* before accessing an extendible array.
`lock`($array\_name, lock\_mode$) locks all of the extendible array storage, or a portion of the host array storage bound to a sharing array. $array\_name$ denotes the name of the extendible array or the sharing array to be locked. $lock\_mode$ is one of the following three:

- READ: The storage for the array is read locked. This lock is shared by any other clients and the storage is not writable nor extendable by any clients.
- WRITE: The storage for the array is write locked. It is not extendable by the client that aquires this lock. Also it is not readable, writable nor extendable by other clients.
- DYNAMIC: If one of the clients extending or reducing an extendible array after acquiring this lock, the other clients cannot extend nor reduce the extendible array nor access any elements. This lock is applied only to extendible arrays and not to sharing arrays.

The following should be noted.

- The *lock_mode* must compatible with the permission specified at the creation of the extendible array. For example, if a client process has not '`x`' and '`w`' permissions, DYNAMIC and WRITE locks are not possible.
- Locking of the extended part of a sharing array is not necessary, since the extension of a sharing array is local to the client process and cannot be accessed from any other processes.

Example 5 The following program references to an extendible array `ea` on the server side. After a sharing array `a` shares all of the storage area of `ea`, it extends its size relatively to the current size. The extension is allocated on the client side.

```
extern public int ea[][];
   /* declare the reference to 'ea' on the server side*/
main()
{
   int  a[:][:];             /* declare a sharing array */
   int  mark, i, j;

   share(ea, a, REDUCE, 0, bound(ea,1), 0, bound(ea,2));
      /* bind 'a' to the host array 'ea' */
   mark = bound(a,0);        /* memorize the current shape of 'a' */
   extend(a, REL, 10, 10);   /* extend locally to this client */
   lock(a, READ);            /* READ lock on the shared part of 'a'*/
   ......................    /* working */
   for (i=0; i< bound(a,1); i++)   /* print all elements of 'a' */
     for (j=0; j<bound(a,2); j++)
       printf("%d ",a[i][j]);
   reduce(a, mark);          /* reduce to the size at sharing */
   unlock(a);                /* unlock 'a' */
   unshare(ea,a);            /* unbind 'a' */
}
```

6 Conclusion

We have described a distributed system for scientific applications based on sharable extendible arrays. After proposing some properties of extendible arrays, we described some important design issues stressing sharing schemes of

extendible arrays. The prototype of the system has been constructed and is running on the UNIX distributed environment. It currently serves various numerical scientific applications employing huge persistent multi-dimensional arrays.

Our future works include the followings:

1. design and implementation of a transaction system for the processing of extendible arrays and sharing arrays
2. redesign of our extendible array system as a platform of multidimensional OLAP(MOLAP) system, in which, unlike the traditional ones, incremental processing is possible against data increase

Our future works include design and implementation of a transaction system for the processing of extendible arrays and sharing arrays.

References

1. K.E.Seamons and M.Winslett, 'Physical Schemas for Large Multidimensional Arrays in Scientific Computing Applications', Proc. of 7-th International Working Conference on Scientific and Statistical Database Management, 218-227, 1994.
2. S.Sarawagi and M.Stonebraker, 'Efficient Organization of Large Multidimendional Arrays', Proc. of International Conference on Data Engineering, 328–336,1994.
3. H.Gupta, V.Harinarayan, A.Rajaraman, and J.D.Ullman, 'Index Selection for OLAP', Proc. of 13-th International Conference on Data Engineering, 208-219, 1997.
4. Y.Zhao, K.Ramasamy, K.Tufte and J.L.Zhao, 'Array-Based Evaluation of Multi-Dimensional Queries in Object-Relational Database Systems', Proc. of 14-th International Conference on Data Engineering, 241-249, 1998.
5. A.P.Marathe and K.Salem 'A Language for Manipulating Arrays', Proc. of 24-th International Conference on Very Large Databases, 241-249, 1997.
6. N.Widmann, and P.Baumann, 'Efficient Execution of Operations in a DBMS for Multidimensional Arrays', Proc. of 10-th International Working Conference on Scientific and Statistical Database Management, 155-165, 1998.
7. A.L.Rosenberg, 'Allocating Storage for Extendible Arrays', JACM, Vol.21, 652-670, 1974.
8. A.L.Rosenberg and L.J.Stockmeyer, 'Hashing Schemes for Extendible Arrays', JACM, Vol.24, 199-221, 1977.
9. Niklaus Wirth, 'Programming in Modula-2', Springer-Verlag, 1983.
10. E.J.Otoo and T.H.Merrett, 'A Storage Scheme for Extendible Arrays', Computing, Vol.31, 1-9, 1983.
11. A.Novacek, 'Using Time Stamps for Storing and Addressing Extendible Arrays', Computing, Vol.37, 303-313, 1986.
12. T.Tsuji, M.Iguchi, and K.Watanabe, 'An Implementation Scheme of Extendible Arrays and Its Application to Pascal Language', Transactions of Information Processing Society of Japan, Vol.37, 843-853, 1988.

Pini – A Jini-Like Plug&Play Technology for the KVM/CLDC

Dipl.-Inform. Steffen Deter and Dipl.-Inform. Karsten Sohr

University of Marburg
Department of Mathematics and Computer Science
deter,sohr@mathematik.uni-marburg.de

Abstract. The Jini™ technology provides an infrastructure for spontaneous networking of devices based on Java-classes and - interfaces. With the help of Jini, small devices with minimal resources should be able to join such infrastructures. Due to the fact that Jini is based upon RMI (Remote Method Invocation) and is therefore on top of RMI, it is impossible to Jini-enable these limited devices: the use of RMI by Jini wastes resources which are not available in the aforementioned limited devices. A potential approach to Jini-enable limited devices is to replace the RMI-technology with the RPC-technology (Remote Procedure Call). By means of this technology it is possible to provide an efficient mode of communication for Jini components, i.e. services and their proxies. However it is important to bear in mind, that this technology avoids the major part of resource waste.
A revision of the implementation strategy for this reason and to adapt the Jini technology to the KVM/CLDC is necessary. In the following sections, the integration of an RPC-implementation into the Jini concept and some revised structures of the Jini-implementation strategy will be described.

1 Motivation

The Jini technology [1, 2] which is based upon Java-classes and interfaces [3], provides an infrastructure for spontaneous networking of devices. These devices can be either a service provider or a service consumer. As mentioned in the abstract, often only minimal resources are available for these devices. Yet in this particular field Jini shows great problems: The necessary Jini core components provided by the package "net.jini.core" have only limited functionality. The dynamic spontaneous networking may only be provided by the combination of this core package with the Jini-extension packages. However these extension packages are major resource consumers. Furthermore, the use of the RMI-technology [4] increases the need of resources. Therefore the major goal is a small and efficient implementation of Jini. It is of great importance to avoid using the RMI-technology, yet it is necessary to provide dynamic and efficient spontaneous networking for limited devices.

To achieve this goal, we intend to provide a Jini-like technology – called Pini – that runs on the KVM/CLDC [10].

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 53–66, 2001.

The testing ground of this implementation will be the PABADIS project (Plant Automation Based on Distributed Systems)[5]: The field of plant automation provides interesting case studies to demonstrate the effect of joining network infrastructures by means of spontaneous networking and agent technology. On this testing ground often only limited devices and/or platforms are available. The term "devices" refers to hardware, which often provides only limited resources in the sense of memory, storage, computational performance, etc. Platform means the available Java platform, e.g., the available JDK version is often less than the JDK 1.2, which is required by Jini [2].

The following sections provide a description of the Pini approach that enables limited devices to use a Jini-like technology for joining networks in a spontaneous manner.

2 Jini Key Components Adapted to Pini

This section provides an overview of some Jini components which have been reimplemented and/or adapted to the Pini-context. Specifically, these components are the discovery and join protocol, the Pini lookup service and its (Pini) lookup service proxy.

2.1 The Pini-Discovery and the Pini-Join Protocol

To participate in a Jini community Jini services and clients have to discover **at least** one lookup service in order to register themselves or to get information about available services in that community (in case of both clients and services). This discovery feature is provided by the discovery protocol. As result of a successful discovery process the service or client performing discovery receives a lookup service proxy. In Sun's lookup implementation this proxy object is submitted as a RMI-marshalled object, and the proxy class will be loaded by the RMI class loader[4]. In the Pini implementation the discovering service/client does not receive a RMI-marshalled object (because RMI is not available), but receives a *ServiceDescription* (see Section "3.3 The Class ServiceDescription") via TCP/UDP, which contains all the necessary information to retrieve the proxy. This essential information contains the URL of the proxy classes and the initial data for this proxy. Now the proxy classes will be loaded by the "Pini class loader" (for details see Section "3.5 The Pini Class Loader") and initialized with the submitted initial data.

To perform discovery on limited devices some modifications of the Jini discovery protocol (as described in the Jini Specification) are necessary: Since it is intended to run Pini on the KVM it is impossible to use multicast for discovery (there is no multicast feature available on the KVM). However, to provide a way for discovering lookup services without any knowledge or preconfiguration the discovery protocol uses broadcast instead. The following figure describes the discovery process again: The client/service starts the discovery process, so that discovery requests are sent out (either via broadcast/multicast or unicast). The

lookup service receives these requests and responds with the required information, particularly the URL of the proxy classes and initial data. This information is encapsulated within an object of type *ServiceDescription* (see Section "3.3 The Class ServiceDescription") which can be seen as a kind of "marshaled" (proxy) object. After receiving the response, a discovery event is generated and delivered to the waiting *DiscoveryListeners.*

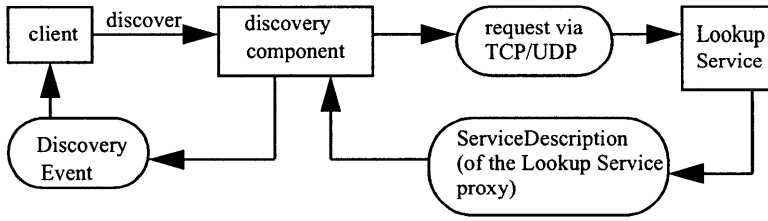


Fig. 1. The (adapted) discovery process in Pini

With the help of the aforementioned lookup service proxies, the hosts (clients/services) can communicate to the (discovered) lookup services, for instance, to register services or to perform lookup for available services registered at these lookup services. In Sun's lookup service implementation the communication between lookups and their proxies is strongly based upon RMI and the lookup service itself is registered as a RMI remote object at the RMI-Registry.

Such a registration at any *registry* is unnecessary in Pini, because every (Pini-) lookup service proxy knows its "home". The lookup service proxy only has to know the IP-address of its lookup service and the appropriate port number for the communication.

2.1.1 Lookup of Services

The process of looking up services is implemented by calling the `lookup(..)`-methods of the lookup service proxy. Within the proxy implemented by Sun, however, the remote `lookup(..)`-methods of the lookup service are called via RMI, and the services found are returned as RMI-marshaled objects. After extracting the URL of the proxy classes from the marshalled object, the classes of the found services are loaded via the RMI class loader.

In the Pini implementation the lookup service proxy also calls the remote `lookup(..)`-methods of the lookup service, but not via RMI. The lookup service proxy submits the search patterns via TCP/UDP to the lookup-service (as a remote procedure call of the appropriate remote methods). After receiving this remote method call request, the lookup service searches for matching services and resubmits the *ServiceDescriptions* of these found services back to the requesting host. This host generates *ServiceItem* objects, and if the method `lookup(ServiceTemplate, int)` was called, the generated *ServiceItems* are encapsulated in a *ServiceMatches* object. Now, the search result will be returned to

the requesting client/service either as the service proxy of the found service (in case of invocation of method `lookup(ServiceTemplate)`) or as a *ServiceMatches* object (in case of invocation of method `lookup(ServiceTemplate, int)`).

Figure 2 depicts this process: A client/service calls a `lookup(..)`-method at the lookup service proxy and provides the search patterns. The proxy submits this information to its lookup service as a remote method call request via TCP/UDP. The search results are returned by the lookup service via TCP/UDP back to the proxy, which generates either the service proxy or a *ServiceMatches* object.

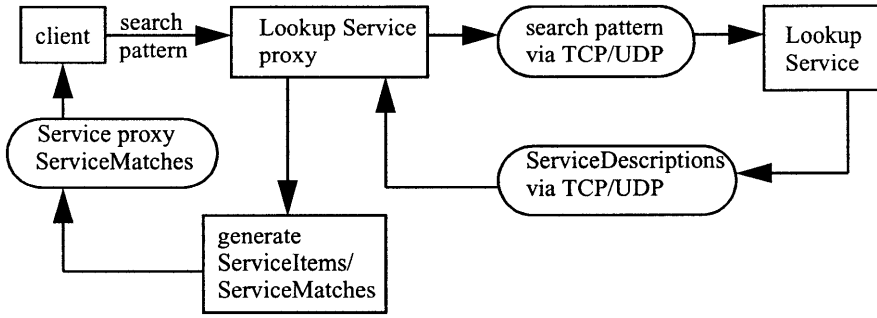


Fig. 2. Lookup for services

2.1.2 The Registration of Pini-Services at Pini-Lookups

To be available within a Jini community, services must be registered with lookup services (at least with one) by calling the `register(...)`-method of the lookup service proxy. Within Sun's implementation of the lookup service proxy the remote `register(...)`-method of the lookup service is called via RMI.

In Pini the registration data (*ServiceID*, *ServiceDescription*, attributes, lease duration) are sent via TCP/UDP to the lookup service as a remote procedure call. A further difference between Jini and Pini is: Instead of the *ServiceItem* which is passed to the `register(...)`-method as a parameter, the member fields of that *ServiceItem* are sent to the Pini lookup service. Strictly speaking, a *ServiceDescription* containing all the aforementioned member fields and instead of the service proxy the codebase URL, initial data, etc. will be sent to the lookup. This *ServiceDescription* will be generated by the lookup service proxy via the method `generateDescription()` during the register process. The lookup service receives this registration data and registers the service. As the result of a successful registration process, the lookup service returns the granted lease and the *ServiceID* as a *ServiceRegistration* object back to the lookup service proxy (via TCP/UDP). The proxy in turn returns it to the requesting service.

Figure 3 shows the individual steps of the registration process:

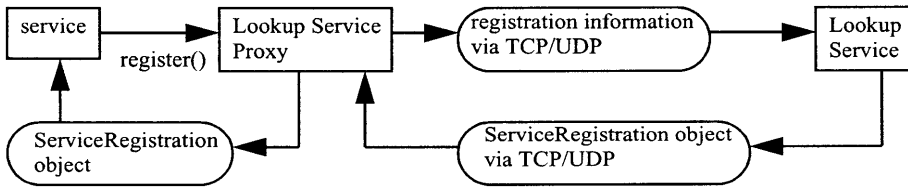


Fig. 3. Registration of a service at a Pini lookup service

2.2 The Pini-Lookup-Service

Lookup services are very important for Jini- and Pini infrastructures, therefore it is necessary to have at least one lookup in every Jini/Pini community. Hence, the resource restrictions of limited devices greatly influence such lookup services also. Yet, the functionality must not be reduced.

A common way to reduce the resource consumption of lookup services is to integrate the RPC-mechanism into the communication between lookups and their proxies. Thus it is possible to avoid the use of RMI for that communication: In Sun's Jini implementation the communication between lookups and their proxies is strongly based upon RMI. Therefore lookup services are registered as RMI-remote objects at the RMI-registry.

By avoiding the use of RMI in the Pini approach, a (Pini-) lookup service need not be registered at a registry like the RMI registry. The entire communication between lookups and their proxies is based upon a Pini-RPC implementation. For details about the protocol see Section "3.4 The Communicator-Package". Moreover, all necessary service-side functionality is described in Section "3.2 Pini-Services" in more detail.

This section only shows the overall integration of RPC into lookup services. Therefore, figure 4 gives the concept of this integration: A lookup service proxy sends a method call request to the lookup. It is then necessary for (lookup service) proxies to know the IP address and the port number of the appropriate lookup. The lookup services will receive the request via the RPC-implementation of the communicator package. This request is decoded by the RPC-mechanism, and the appropriate method within the lookup service will be invoked. The result of such a method invocation is resubmitted to the method caller with the help of the communicator package implementation (exactly: the class *ResultDelivery* is responsible for resubmitting results back to clients).

Examples for communication between lookups and their proxies are: the process of looking up services available in a community (Section "2.1.1 Lookup for Services") or the registration of a service at lookup services (see Section "2.1.2 The Registration of Pini-Service at Pini-Lookups").

2.3 The Lookup-Service Proxy

The main functionality of lookup service proxies is provided by the *ServiceRegistrar* interface of the Jini-specification, however this functionality must not be

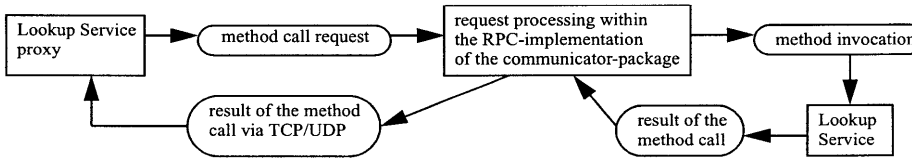


Fig. 4. Communication between a (Pini) lookup service and a lookup service proxy via RPC

reduced. Thus, the major concern here is to optimise the communication between lookup service and proxy. This means that the RPC-mechanism has to be integrated into the communication instead of RMI.

Sun's reference implementation of the lookup service proxy is strongly based upon the RMI-technology. That means that remote methods are invoked via RMI.

In the Pini implementation, however, remote method calls are realized via RPC: the proxy sends out method call requests via TCP/UDP and listens for return values based upon socket servers. These abilities are provided by the communicator package, specifically by the class *Communicator* of this package. To resubmit the result to the requesting lookup service proxy, the lookup service puts it into an object-array. Thus, it is the task of the lookup service proxy to extract the correct return values from that array. An example of the return values extraction from the received object-array is the generation of *ServiceItems/ServiceMatches* within the `lookup(..)` methods of the lookup service proxy (see Section "2.1.1 Lookup for Services"). Therefore the proxy reads the items in the array, casts them into their appropriate types, generates the return values and delivers them to its client. Further details are described in Section "3.1 Service-Proxies and the Class DefaultPiniProxy" and Section "3.4 The Communicator-Package".

Another important modification of the lookup service proxy structure is the extension of the class *DefaultPiniProxy*. This is due to the fact that the proxy needs to know the IP-address of its lookup service and its port for the (remote) communication. By this extension the lookup service proxy inherits the following member fields:

```
public String serviceAddress
public int servicePort
public String codebase
public Class[] interfaces
```

These fields must be initialized by the inherited method `initialize(String, int, String, Class[])` (See Section "3.1 Service-Proxies and the Class DefaultPiniProxy").

3 New Components and Modified Jini Interfaces

This section describes components, that belong to Pini but not to Jini. Moreover, a description of new features that (Pini-) services and proxies require in order to provide the basics for communication via RPC are described. One of these new features is the extension of the class *DefaultPiniProxy* by (Pini-) service-proxies as described in the following section.

The new components include the communicator package "pabadis.kvm.pini.communicator" and the class loader in "pabadis.kvm.pini.util".

3.1 Service-Proxies and the Class DefaultPiniProxy

Service proxies provide their services to clients. The provision of services to clients occurs either with or without communication between the proxies and their services. It is noteworthy to mention, that this communication often occurs via RMI. In Pini the use of RMI technology shall be avoided so that the necessary abilities are provided by the package "pabadis.kvm.pini.communicator". For remote method calls, the method `callMethod(..)` is provided by the class *Communicator*. As parameters this method expects the name of the (remote) method to be called, represented as a string, and an object array containing the appropriate (remote) method parameters. Therefore a method call from a proxy to a service looks as follows:

```
Communicator com = new Communicator(String,int);
Object[] result=com.callMethod(String, Object[]);
```

As the result of the method invocation, the *Communicator* returns an object array containing the necessary information to generate the correct return value(s). The proxy then reads the items from the array, casts them to their appropriate types and generates the expected return value. To obtain the correct value, the communication partners need an appropriate communication scheme. This means that the object in the array should be ordered in a well-known fashion, so both the proxy and the service are able to interpret the value for the method call request. Furthermore, the generation of the appropriate return value from the returned object array must be done by the service proxy; it does not happen automatically!

Another new feature is that **all** service proxies have to extend the class *DefaultPiniProxy*. Hence all service proxies inherit the following fields:

```
public String serviceAddress
public int servicePort
public String codebase
public Class[] interfaces
```

These fields must initialized by the inherited method `initialize(String, int, String, Class[])`. Through the first `String` parameter, the IP-address of the appropriate service is provided to the service proxy. By means of the `int` parameter, the port number will be provided to the service proxy. The second `String` parameter represents the codebase URL of the proxy class file. Finally, the `Class`-array contains the appropriate interfaces, that this proxy implements. To ensure that the array does not contain interfaces, which the proxy does not implement, the lookup service proxy checks this array during the register process. Remember, these array items will be compared to the type values of a given ServiceTemplate during the lookup process.

One may be inclined to inquire, why this is. On the one hand, the extension of the class *DefaultPiniProxy* by Pini service proxies provides a common interface for initializing the downloaded proxy classes, or more concisely, the instance of this particular proxy class. On the other hand, the inherited fields provide the necessary information for using the RPC-protocol, for downloading the class files and for performing lookup. Therefore, this means that the "main" proxy class needs to extend the *DefaultPiniProxy* class, yet not other additional proxy-classes.

3.2 Pini-Services

This section describes the necessary service-side features for the communication between services and their (remote) proxies. If "unlimited" resources are available, the use of RMI is a favourable way to enable remote communication. In the case of limited devices however RMI is not the best solution due to the fact that it is a huge resource consumer.

In Pini an RMI-like registration (at the RMI-registry for example) is unnecessary. To provide remote communication the service must merely ensure that its service proxy knows the IP-address and the port number for communication. This necessary information is put into the *ServiceDescription* of that service. Upon registering the service with lookup services, the *ServiceDescription* is sent to the lookup instead of the whole service proxy. Hence, if clients download the *ServiceDescription*, they obtain all necessary information to load the proxy classes, to initialize the proxy and to communicate back to the service via the proxy. (For *ServiceDescription* see Section "3.3 The Class ServiceDescription", for service registration see Section "2.1.2 The Registration of Pini-Services at Pini-Lookups").

The communication between services and proxies is based upon TCP/UDP. An appropriate mechanism must therefore be provided for such communication. Since the major part of the remote communication between services and proxies is the invocation of remote methods, the following explanation only refers to this interesting field: Specifically, this section describes the necessary service-side conditions for using the (Pini-) RPC-implementation. The integration of RPC into services containing the reception and processing mechanism for method call requests, the method invocation and the resubmitting of the results back to the clients will be highlighted in this section.

As described in Section "2.2 The Pini-Lookup Service", the **reception and processing** of the requests are provided by the classes of the communicator-package: An instance of class *MethodCallListener* listens for incoming requests and generates *MethodCallEvents*. These events are delivered to a *MethodCall-EventHandler* instance. Within this event handler_instance the events are **processed** and the appropriate **methods are called**. The results of these method invocations are **resubmitted** to the clients via class *ResultDelivery*.

Note, that the implementation of the *MethodCallEventHandler* must be done by the programmer. All other necessary features are provided by the package "pabadis.kvm.pini.communicator" (For more details about this package see Chapter "3.4 The Communicator Package")

3.3 The Class ServiceDescription

This class contains not only all necessary information to register the proxy at Pini lookup services, but also information necessary to load and initialize the proxy-object on the client side. In particular, this information includes the service address (IP-address), the service port number, the URL of the proxy classes, the *ServiceID* of the service, the name of the main proxy class, initial data and an array of interfaces that the proxy implements.

The data on the lookup service side are used to find matching services when a `lookup(..)`-method call occurs. Therefore, these data are the initial base for comparison.

On the client side, these data are conversely necessary in order to retrieve the proxy object: The proxy classes are therefore loaded from the URL. The class will then be initialised with the initial data. This action occurs within method `getServiceItem()` which returns a *ServiceItem* object, which contains the proxy-object.

3.4 The Communicator Package

The communicator package provides all necessary classes for the RPC-communication between Pini-services and their proxies. As described in the aforementioned sections, there is a demand for mechanisms to exchange method call requests, and to receive the results. These data exchanges contain the receiving and sending data at both the service proxy side and the service side based upon TCP or UDP. However, there are different mechanisms for services and proxies: The class "Communicator" provides all necessary functionality for sending method call requests and receiving the results (generally speaking at proxy side). Meanwhile, the classes *MethodCallListener*, *MethodCallEvent*, *ResultDelivery*, and the interface *MethodCallEventHandler* provide all necessary functionality for receiving method call requests and sending the results of those method invocations:

3.4.1 The Class Communicator

As previously mentioned, this class provides all necessary functionality for RPC-communication of proxies to their services. This refers to the sending of method

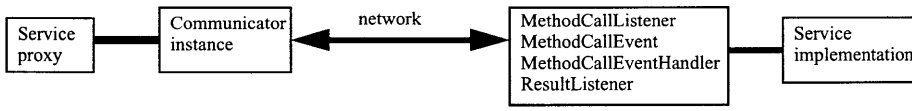


Fig. 5. Overview over the communicator-package

call requests and receiving of results. Only the constructor of the class `Communicator(String, int)` and the method `Object[] callMethod(String, Object[])` exhibit a public interface. For the initialization of this class, the above constructor expects a string representation of the service's IP-address and the appropriate port. On invocation of the method `callMethod(..)` this information is used to establish the communication connection. As parameters the method `callMethod(String, Object[])` expects a string representation of the method identifier and an object array containing the parameters of the method to be called. Within the method `callMethod(..)` a thread will be created that is responsible for receiving the result of the method call request. As the next step, a communication connection to the service will be created (to the listening socket server at service side within a *MethodCallListener* instance) based upon the above information (IP-address and port). With the help of this connection the method identifier, the return port (on which the above thread listens for the result), the length of the parameter array and then the parameters are sent to the service. After receiving the result (through the above thread) it will be returned to the client as an object array. The client is then responsible for generating the appropriate return value from this information. (see Section "3.1 Service Proxies and Class DefaultPiniProxy").

3.4.2 The Class MethodCallListener

The class MethodCallListener is used on the service side to receive remote method call requests. This means that data is received via TCP/UDP-connections. For this reason the class *MethodCallListener* has a thread that listens for incoming method call requests (see previous section), which are buffered in a queue. Thus, the further reception of incoming requests will not be blocked by the processing of such received requests. Processing the buffered requests is done by another thread, named the *ParserThread*. The *ParserThread* determines the source IP-address (for return of results) of the request, reads the return port number (of the listening thread at client side), the method identifier, the number of the parameters and the parameters as objects. With this information a *MethodCallEvent* is generated and passed to the instance of the *MethodCallEventHandler* (which is passed to the *MethodCallListener* as a constructor parameter). After the result is retrieved from the event handler, it is passed to an instance of class *ResultDelivery*, and the results are sent back to requesting client. (Hence every method call request is sent in a "secure" manner because every method call request has a result. If not, the method caller will be informed via a RemoteException!)

Specifically, the class *MethodCallListener* has only two public methods/ constructors: the method `int getPortNumber()` to retrieve the port number (on which the *MethodCallListener* listens for incoming method call requests). The second is the constructor `MethodCallListener(MethodCallEvent-Handler)`. The single argument of the constructor is the event handler that processes the generated *MethodCallEvents*. For an overview of the functionality of the *MethodCallListener* see the figure below:

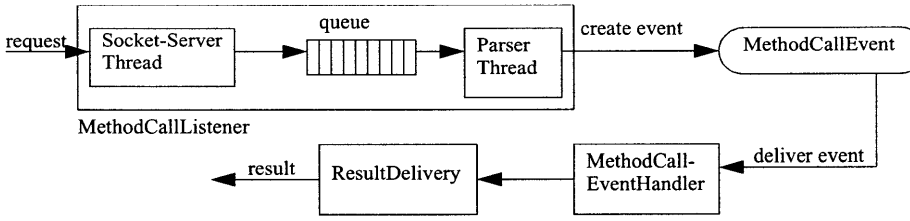


Fig. 6. Overview over the MethodCallListener-class and its functionality

3.4.3 The Class MethodCallEvent

A MethodCallEvent contains all necessary information pertaining to a method call request. In particular, the class contains the following member fields:

```
public class MethodCallEvent{
  private String methodName;
  private Object[] parameters;
  public String getMethodName();
  public Object[] getParameters();
  public MethodCallEvent(String, Object[]);
}
```

It is important to note that this class is not an extension of class java.util.Event-Object, because this package may not be available on limited devices. However, the basics for event processing in the Pini-concept are the same as in the Java event model.

It is not necessary to look at the member fields in detail. As mentioned before, those *MethodCallEvents* are generated within the *ParserThread* of class *Method-CallListener* and passed to the *MethodCallEventHandler*. This event handler is the subject of the next section.

3.4.4 The Interface MethodCallEventHandler

The major task of the *MethodCallEventHandler* is the processing of *MethodCall-Events* that are generated by the *ParserThread* of the *MethodCallEventListener*.

For this event processing the interface contains only a single method: `Object[] parseEvent (MethodCallEvent)`. However, the implementation of this method is up to the service programmer. Hence the information within the event must be interpreted, and the appropriate actions must be implemented: the determination of the method to be called and which parameters this method expects. The parameters are provided by method `Object[] getParameters()` of the class *MethodCallEvent*. After a type cast to the appropriate type the parameters can be passed to the method that has to be invoked. In order to return the result to the client, an object array will be generated and passed to the *ResultDelivery* class. This class is responsible for sending back the result array to the method caller.

3.4.5 The Class ResultDelivery

The primary task of this class is to send the results of a remote method invocation back to the appropriate client. Since the resubmission of results is independent from any other action, the *ResultDelivery* class is an extension of class Thread. Due to this, the processing of further *MethodCallEvents* by the *MethodCallEventHandler* is not blocked by the resubmitting of those results. As mentioned in Section "3.4.2 The Class MethodCallListener", the result of the method call will be passed to the *ResultDelivery* instance and sent to the appropriate client.

Therefore, a communication connection is established via TCP or UDP, which is based upon the IP-address and the port number passed to the constructor `ResultDelivery(String, int)` of this class. The result array is sent to the appropriate client through this connection. After that the thread can be destroyed.

3.5 The Pini Class Loader

Class loaders dynamically load classes if necessary. URL class loaders in particular download classes from a specified URL. The Pini class loader also functions in this manner: In particular, the Pini class loader has to download the class files and write them to a temporary classpath. After that, the appropriate `Class`-object will be created via `Class.forName(name)`. Thus, the *PiniURLClassLoader* is akin to a class FILE loader.

Implementing such a class loader is necessary because the KVM does not provide user defined class loaders, however proxy classes must be downloaded from their codebase. When the client shuts down or starts up the computer, all classes in the temporary classpath are deleted.

4 State of the Art and Results

This section descibes the current state of the art of the Pini technology and some results in comparison to Jini.

At the time of writing this paper the following features are available. The lookup service is implemented and works fine. Services and clients can perform

Table 1.

Jini core	32 kB
Jini extension	230 kB
Lookup Service	501 kB
RMI	401 kB
Summary	**1167 kB**

discovery and lookup. Services can register themselves with lookup services, leases will be granted by the lookup service for service and remote event registration. Additionally, the mechanisms for using attributes and adminstration are currently in development.

Table 1 shows the Jini disk space cosumption. In comparision to Jini it is noteworthy that the memory consumption of Pini is much lower: currently Pini needs about 250 kB and there are no other features necessary.

The run time memory consumption of Pini should be determined in the next few months. The comparision of this memory consumption to Jini will be explained in further publications.

5 Related Work

As mentioned in the sections before, the major resource consumer within Jini is the RMI-technology. Therefore, our major goal was to avoid this resource waste and thus to prevent the use of RMI. However, there are several attempts to provide so-called RMI-light technologies, e.g., [8]. Hence, on the one hand it may be possible to replace Sun's RMI technology used within Jini with such a RMI-light technology in order to Jini-enable limited devices. On the other hand, these aforementioned technologies sometimes use Java features that are often not available on limited devices.

An attempt to provide a Jini-like technology for limited devices is described in [6]. But unfortunately, this technology also uses features that are mostly not available on limited devices.

Sun has realized that it is impossible to Jini-enable limited devices. Therefore they recommend their "Surrogate Architecture"[9]. However, this technology cannot be used as a stand-alone Plug&Play technology: It expects a proxy host (the so-called "surrogate host"), which connects the device to the "Jini world".

A very similar technology to Pini is the TINI technology, which was presented on the "Fifth Jini Community Meeting" in December 2000 [7]. The advantage of this technology over the aforementioned technologies is that TINI works as a stand-alone technology and is available for the KVM/CDC (Kilo Virtual Machine with the Connected Device Configuration). However, in comparison to this Pini is available for the KVM/CLDC (Kilo Virtual Machine with the Connected **Limited** Device Configuration), which has a much more limited functionality than the KVM/CDC.

6 Conclusion and Future Work

This paper demonstrates the possibility of implementing a Jini-like technology that is simple, small and uses RPC instead of RMI. Moreover, spontaneous networking of limited devices may be possible with such a technology. The communicator package provides a simple and efficient way for remote method calls and hence, efficient communication between services and their proxies. In addition, the network traffic can be reduced by submitting only *ServiceDescriptions* instead of full service proxy objects to the clients.

Another measure which would provide more comfort in using Pini, is the development of a compiler, which is able to generate classes like RMI-stubs and skeletons in order to use the object-oriented programming model in the remote context.

Furthermore, we want to make a tool available to connect the Pini technology to the Jini technology. This means, that we will provide a technology, which makes it possible to use Jini services within a Pini community, but also to use Pini services within a Jini community.

References

1. W. Keith Edwards; 1999: **Core Jini**; Prentice Hall PTR; ISBN 0-13-014469-X
2. K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, A. Wollrath; 1999: **The Jini™ Specification**; Addison Wesley; ISBN 0-201-61634-3
3. J. Gosling, B. Joy, G. Steele, G. Bracha; 2000: **The Java™ Language Specification**; Addison-Wesley; ISBN: 0201310082
4. Sun Microsystems: **Java™ Remote Method Invocation (RMI) Specification**
   http://java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html
5. Pabadis Consortium: **http://www.pabadis.org**
6. Stephan Preuß; University of Rostock; 2000: **NetObjects Dynamische Proxy-Architektur für Jini** in "NetObjectDays 2000" work proceedings
7. Alan Kaminsky, Rochester Institute of Technology, 2000: **Running Jini Network Technology in Small Place**, presentation on the "Fifth Jini Community Meeting" http://www.jini.org/jini5/slides/Small_Places/sld001.htm
8. Ch. Nester, M. Philippsen, B. Haumacher; University of Karlsruhe: **Effizients RMI für Java** JIT'99 Java-Informations-Tage 1999; Springer-Verlag ISBN 3-540-66464-5
9. Sun Mircosystems: **Jini™ Technology Surrogate Architecture Specification (Version 0.4)**
10. Sun Microsystems: **Connected Limited Device Configuration, Version 1.0**

Discovering Internet Services: Integrating Intelligent Home Automation Systems to Plug and Play Networks

Wolfgang Kastner and Markus Leupold

Technische Universität Wien
Institut für Automation
Treitlstraße 1, A-1040 Wien
k@auto.tuwien.ac.at

Abstract. With the growing connection of networks to the Internet, setup, configuration and usage of electronic devices becomes more and more complex. Two approaches briefly discussed at the beginning of this article minimize the required knowledge of users for administrative tasks and are based upon well known Internet protocols. Afterwards, we show the dynamic integration of a more or less static fieldbus network connecting low-cost sensors, actuators and further controlling hardware into the Internet environment. Hence, we are able to increase the bus devices' functionality and their accessability, especially by means of the Internet.

1 Introduction

Electronic networks have emerged into almost all areas of our daily life. In the past years – mainly due to the advent of the Internet – the clear distinctions and borders between the different networks started to blur. Today, everyone wants to use Internet-enabled devices regardless which network they originally were designed. As example, Figure 1 shows four areas of our daily life with some appliances used from early in the morn until late night.

In the scientific community the breakthrough application for the Internet was electronic mail. In the area of office and business computing the deployment of the Worldwide Web ignited the final revolution into global networking. In parallel to scientific, office and business networks, the telecommunication networks evolved. While being independent entities used by people around the world to communicate in the past, the introduction of digital telephony in the late 80' and the early 90' was the starting point for more intelligent phones and resulted in end-devices integrating communication facilities directly into office- and home appliances. Regarding modern telecommunication, mobile phones with the Wireless Application Protocol (WAP-phones) implemented are state of the art. Keep in mind, that the WAP protocols are mainly based on "old fashioned" Internet protocols, however optimized for mobile users with wireless devices. Besides gadgets for telecommunication and office networks, consumer electronics devices that initially were designed as stand alone tools like TV, VCR, cameras, etc.,

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 67–78, 2001.

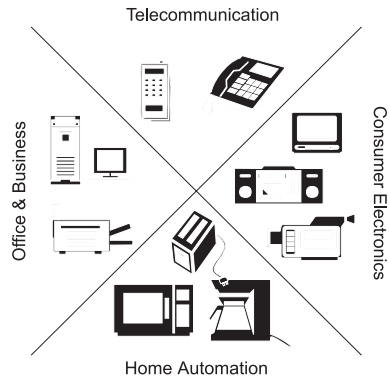


Fig. 1. Classic areas of networking

were combined to proprietary networks. In the present some of these consumer electronic networks are integrated into the Internet, too, because once done, they are more fun to use.

With all this increasing amount of connectivity, a lot of users are overwhelmed by functionality and required knowledge. Devices have to be configured and maintained with more parameters and details than ever. Therefore, industry has turned to create intelligent devices, that do the interconnection work automatically and do not scare away consumers with their functionalities. Two examples of such resulting dynamic networks are shortly discussed in Section 2 and Section 3.

Networks for the home and building area are a rather young development. Remote access to lights and heating or alarm systems fall into this area, as well as kitchen items that speak to each other. Section 4 shortly introduces one network suitable for the home and building area.

Finally, we present how a home automation network can be dynamically linked to the Internet such that devices on each side may benefit from each other. A simple gateway to the home and building devices accesses sensors and actuators, giving other Internet enabled services a way to physically interact with them. In addition, an advanced solution allows to integrate the internal logic of the home and building network by providing a mechanism called binding in both worlds.

2 Universal Plug and Play (UPnP)

In 1999, a group of companies and individuals formed the Universal Plug and Play (UPnP) Forum [13], with the goal, to enable the emergence of easily connectible devices and to simplify the implementation of networks in home and corporate environments. To achieve this goal, the forum published UPnP device control protocols, built on the open, Internet-based communication standards TCP/IP [8] [9], HTTP [5] and XML [3].

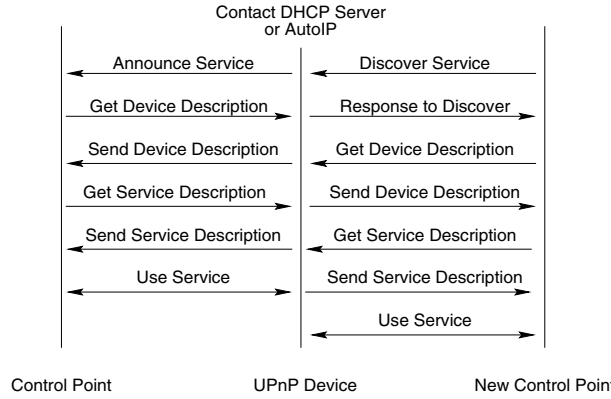


Fig. 2. UPnP-protocols

UPnP provides an architecture of peer-to-peer networking of PCs, intelligent appliances and wireless devices using protocols for *addressing*, *discovery*, *description*, *control*, *eventing* and *presentation*. The UPnP protocols are shortly summarized in Figure 2 and the following paragraphs.

Each UPnP device has to have its own IP address. There are two ways to obtain it: using the *Dynamic Host Configuration Protocol* (DHCP [1]), or, if this service is unavailable, *Auto-IP*. Once available on a UPnP network, each device has to advertise its functionality to the network. Therefore, it has to communicate with so called *control points*, devices that manage other devices and publish them for use. An *advertisement* protocol is used by a device to make its features known to the control points and so to the network. Messages containing the device's URL are sent to a well-known address and port via UDP over IP, being connection-less and without acknowledge. On the other hand, a *search* protocol using a multicast message is invoked by new control points to find available devices. If a device receives this multicast message and believes it matches the required search pattern – if provided in the package – it resends its original message, again containing its URL. At the very moment a control point is aware of the presence of a device, it is able to get more information about the device's features (i.e., its device description and its service description).

The device description has to be written in XML syntax. Instead of describing the device in detail, only the ID and further URLs for the corresponding service descriptions, controls and events are given in XML tags. The device description is obtained by the control point simply issuing a HTTP `GET` command to the device of choice. Once done, the control point may use the URLs retrieved from the device description to download further service descriptions. Again, XML tags describe the services provided by the device as well as necessary arguments, state variables and their data type, range and event characteristics.

After dealing with device- and service descriptions, a control point knows enough about a device to make use of its functionality. It now can invoke actions

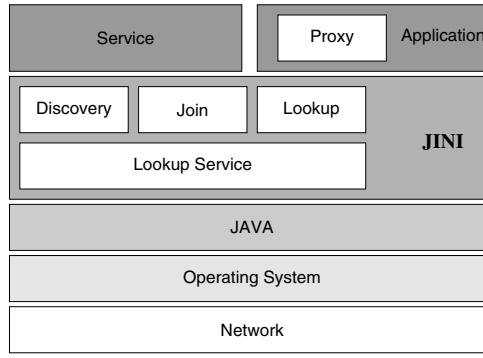


Fig. 3. JINI system overview

and poll state variables to control the device. The most common sequence in communication between control point and device will be:

- The control point sends an action request to the service.
- The service processes the request and returns any results or errors.
- The control point sends a query variable to the service.
- The service returns the value of the proper variable.

To invoke actions, the control point sends an HTTP `POST` message to the service. The Simple Object Access Protocol (SOAP) [2], XML and HTTP may be used for remote procedure call. In addition state variables defined in a service description may be evented. Services that contain such variables publish updates when state changes occur and control points may subscribe to these notifications. A service has to maintain a list of subscribers of its variables. This list contains a *unique subscription identifier*, generated by the service, a *delivery URL*, provided by the subscriber, an *event key* that is increased for each notification to let subscribers know if they missed a message, and a *subscription duration* after which the subscription expires. To subscribe and unsubscribe control points and services again make use of the HTTP protocol's `GET` and `POST` messages.

3 Java Intelligent Network Infrastructure (JINI)

January 1999, when the support for interoperation of applications and devices became more and more important, Sun extended their platform independent programming language JAVA [12] by introducing Java Intelligent Network Infrastructure (JINI) [10]. JINI technology not only defines a set of protocols for interaction between applications and devices, but also leasing and transaction mechanisms for resilience in a dynamic network environment (Figure 3).

The main entity of interest in a JINI enabled system is the *service*. Applications and devices may exchange services, working together using the same resources. Services can be everything: a printer providing possibilities to bring

text and images to paper, a hard disk saving and restoring data, a high-end computer capable of doing matrix-multiplication very fast or devices typically found in our homes, like a garage door, opening and closing at request.

Generally spoken, a JINI service consists of two parts: the "real" *service part* and a *proxy object*. The service part is running on the device providing it and has not necessarily to be written in JAVA, as long as it does the desired job. The proxy object of the service is its interface to the JINI environment. Written in JAVA, it provides methods for accessing the service's components. The proxy object is imported and run by the application or device that wants to use the service. This way, a service may use the user's resources for its tasks, too. It is even possible for a service to have no processing capabilities at all, using a proxy object that does that work for it. The connection maintained between the proxy object and the service is not specified by JINI. It depends highly on the service, and may use JAVA *Remote Method Invocation* (RMI), TCP/UDP sockets or other, even proprietary, connection types.

For applications to know what services are available in the JINI environment, the *Lookup Service* exists. Also called *Registrar*, it provides the central bootstrapping mechanism. If a new service shows up, it contacts the Lookup Service, using a protocol called *Discovery*, and registers using the protocol *Join*. Clients, that want to use the service, request its proxy object from the Lookup Service using another protocol called *Lookup*.

When a new service wants to register in the local JINI system, it first has to detect the available Registrars. This is done by multicasting a *Discovery* request on the subnet. All Lookup Services receiving this request will answer, providing their addresses. Now, the service has to look through the list of answers it received and decide at which Registrars it should register. Typically, it will do this at all available Registrars, but it may choose only one, or some specialized Registrars. When the service knows where it wants to register, it sends a *Join* request to these Lookup Services, consisting of its proxy object and some descriptive attributes. Each Lookup Service will now add the service to its list of available services. At this very point, the service is available in the JINI environment.

If a client wants to make use of a service, it has to query the Lookup Service to find it. Therefore, it must know the available Registrars, which are detected via another Discovery request. Next, it sends a *Lookup* request to the found Registrars, specifying the type of service desired. If the queried Lookup Service knows one or more matching services, it returns a list of them back to the client, which now can choose one, and request the corresponding proxy object.

When the proxy object has been downloaded and initialized by deserializing runtime information from the service, it can be used by the client to communicate with the service. The Lookup Service has done its job and doesn't interfere any longer (except during leasing and eventing which are due to lack of space not discussed in detail here).

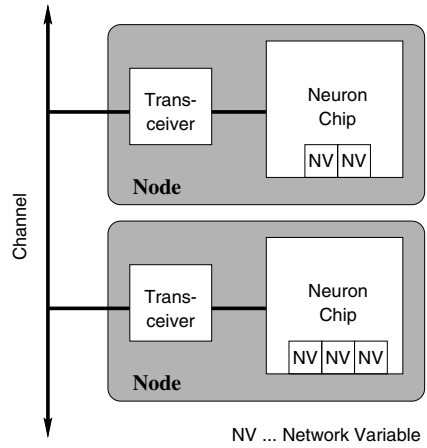


Fig. 4. LonWorks system

4 Local Operating Networks (LONWorks)

Echelon Corporation defined the term *LonWorks system* [11] as a synonym for the fieldbus system Local Operating Networks. Every LonWorks device must make use of a special protocol, called *LonTalk*, for communication. Figure 4 gives a brief overview of a LonWorks system, along with its major components.

The heart of each LonWorks device is the *Neuron Chip*, a system-on-a-chip consisting of RAM, ROM, some I/O and most notably three CPUs. One CPU runs the node-specific program written in Neuron C, an ANSI C dialect that introduces events and task execution order. The other two CPUs are capable of handling the ISO/OSI network layers 1 to 2 and 3 to 6, respectively.

To connect a Neuron Chip to a communication medium (e.g. a twisted pair cable), a *transceiver* is needed. It provides a physical communication device between the chip and the network and both transmits and receives messages. If LonWorks devices use different types of transceivers, they won't be able to communicate directly, because the can't be connected to the same media. This potential problem can be handled by the use of network routers.

As already mentioned, a LonWorks device, called node, is a logical (and often a physical, too) unit consisting of a Neuron Chip, a transceiver, optional memory and I/O conditioning. They exist in many different forms and capabilities, mainly depending on the power of the Neuron Chip. Some are used for data input or output, others are trimmed for control tasks or may contain embedded sensors or actuators. Some of them are (re-)programmable, some have their code stored permanent in ROM. Nodes can be bought off-the-shelf from many different vendors.

The LonWorks protocol is a media-independent protocol. The designer of the system has the choice among a lot of different media and can base the decision on parameters like availability, reliability, performance or security. A *communi-*

cation channel is a specific physical communication medium, to which LonWorks devices are attached by their transceivers. Depending on the medium there can be a maximum of devices per channel, a minimum or maximum distance between devices and different communication bit rates. For example, the common TP/FP-10 is a free-topology, twisted pair channel with 78kbps, 500m maximum distance and up to 128 devices.

In a LonWorks system three kinds of messages can be exchanged: unacknowledged messages, acknowledged messages and repeated messages. If required, messages can be authenticated to prevent unauthorized access to devices. This is done using 48bit keys, which are assigned to the devices at installation time. To simplify data exchange between LonWorks devices, the LonWorks protocol implements the concept of *network variables*.

- *Output network variables*: Some devices obtain certain values from their environment, e.g. a pressure sensor or a button. These values can be queried over the network. Programs running on distant devices may use the network variables like local ones, since the LonWorks firmware transparently does the actual network communication.
- *Input network variables*: Other devices need to be triggered to do their work. For instance, a lamp or a fan have to be turned on when there is need of light or cooling.
- *Binding*: To connect an output network variable with an input network variable a process called binding is used. During network design and installation, the device firmware of an output network variable is configured to know the logical address of a device or group of devices which are interested in its specific piece of data. When the network variable changes, the firmware automatically assembles and sends the proper messages to the devices bound to that variable.

5 Internet Gateway for a Jini-Enabled LONWorks System

The remaining part of this article discusses the approach, to integrate a non-dynamic network into a dynamic Internet networking environment. As shown in Section 4, LONWorks itself has no facilities to be integrated in a self-configuring and self-detecting system. It is hardware-oriented, and is used for physical interaction by the end-user, maintaining actuators and sensors. The dynamic network of choice is JINI, as described in Section 3. It is responsible for the connectivity and helps the user configure the system easily. In [6] we listed some examples for possible functionality enhancements like

- The remote control of the TV set can be used to regulate the heating or control the venetian blinds. The user interface is shown via video text and responses of the fieldbus devices are visualized on the screen.
- When the TV set is turned on, the lights of the appropriate room are dimmed or turned off, depending on user's settings.

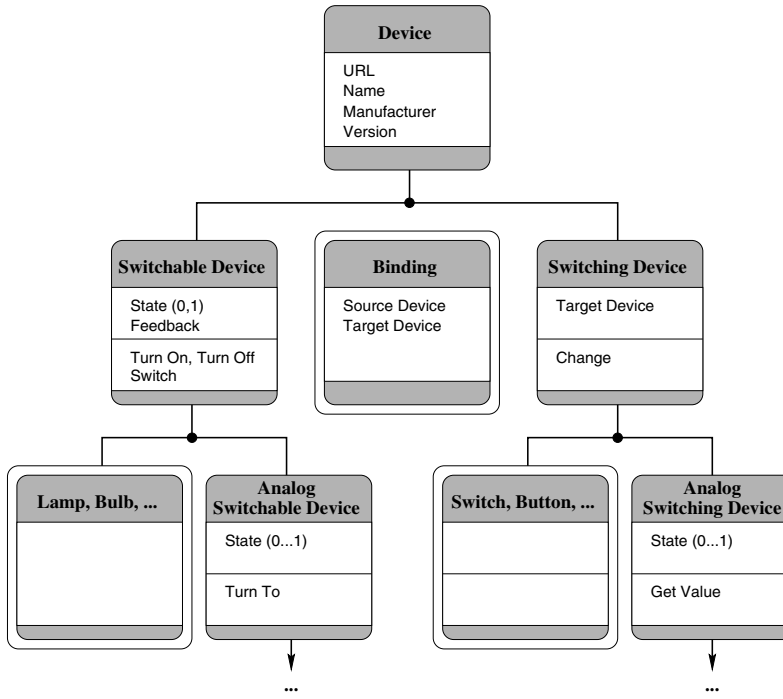


Fig. 5. Class hierarchy

- When a person enters the building (or parts of it, depending on the size of the house) the heating or air conditioning automatically turns on (depending on temperature values, fieldbus devices have recorded).
- and many more...

The increases in functionality mainly result from a connection between consumer electronics and home automation devices produced for LONWorks. For demonstration purpose, and as an example of a practical usage of a JINI-LONWorks interconnection, the environment implemented is the one of a (very) Simple Household System (SHS) depicted in Figure 5.

The root in our hierarchy is made up by the abstract class *device*. It houses generic, documentative attributes that all services and devices must provide:

- their URL for addressing,
- their name as the manufacturer gave them,
- the name of the manufacturer, and
- the version of the device/service implementation.

Except for the URL, the exact notation of the other attributes is not defined in detail. However, all devices have to implement this class. If you look around in peoples homes you will probably find a lot of electronic devices which can be

turned on for use and turned off if no longer needed. In this context, they will be classified as *switchable devices* and have the following features:

- They have an internal state, which can be on or off.
- They can be switched, inverting their state from on to off or from off to on.
- They can be turned on (state is set to on).
- They can be turned off (state is set to off).
- They can feed back commands sent to them to other devices.

In our object hierarchy, the switchable device forms the root for all services that can be triggered. It is an abstract class, and devices have to implement it, if they want to be switchable. These are for instance, simple devices like lamps or bulbs. Of course not all devices in our homes have only two internal states. Many of them have a lot to infinite states between those extremes. For example, a heating element in a stove can be regulated, or a dimmable light can be set to 75% power. Those devices are represented by the abstract class *analog switchable device*. Still being able to be used as switchable device, it extends this class with the following features:

- The internal state is extended to be able to hold any value between zero (off) and one (on).
- It can be turned to a specific value (e.g. 0.75), too.

All services described so far depend on external devices triggering them. The parent of all kinds of switches is the abstract class *switching device*. It gives all switches the following features:

- It stores the reference to the service, which should be triggered. Each switching device can only handle one service at a given time. When the device is switched, the selected service is told so. This feature will be used when the switching device is in charge of triggering something.
- It can be queried if it has been switched. This way, external services can make use of the switching device without the need of the device itself to know how to trigger something. This feature is used when an external device is in charge of triggering something, but uses the switching device for user interaction.

Switches allow the user to choose what device they will trigger. The ability to be able to remotely connect devices can be established by using *binding*. Binding is a service that lets the user – or administrator – choose a switching device and a switchable device. From this point in time it is in the scope of the binding service to make sure that using the switch triggers the proper device. The user should need as less knowledge as possible to interconnect devices. Therefore, the binding checks if the chosen devices can interoperate at all and on what level: if both switch and target device are analog, they are connected that way; if only one device understands analog commands, both devices fall back into a simpler way of communication.

It is also in the scope of the binding object to delegate the task of binding. If the source and target devices are located on the same infrastructure (e.g. a fieldbus), and this infrastructure provides binding, too, it may be invoked by the binding object. On the other hand, if source and target devices are of two completely different infrastructures, the binding object may be in charge of simulating the connection by querying the source and informing the target in the event of a change.

Two approaches with increasing levels of complexity for a Jini-enabled LONWorks system have been under development:

- *Simple Internet gateway.* In this approach, LONWorks is used to physically connect hardware devices like lamps or buttons, but provides no internal logic. Instead, each device is assigned a JAVA application (proxy object) capable of connecting to LONWorks and accessing its network variables. The advantage of this approach is the complete abstraction of LONWorks services to the JINI/Internet world. Only some dedicated proxy objects need to know how to interact with the LONWorks network.
- *Advanced Internet gateway.* It is obvious, that the simple gateway degenerates LONWorks to be just a provider for connections to the proper hardware services. LONWorks-internal logic like the binding mechanism is completely ignored. Therefore, if two LONWorks services interact, no shortcut at the LONWorks level is possible. Hence, a more clever approach is to include LONWorks binding whenever possible, that is when network variables are exchanged in the LONWorks network, but that particular information is not needed anywhere outside LONWorks. In that case, the propagation delay from a LONWorks device to a proxy and back to another LONWorks device can be avoided, thus being a lot faster.
  The simple services presented in the first approach can be improved to examine a user's request for a binding. If two LONWorks devices on the same LONWorks network should be bound, it configures the LONWorks firmware in the corresponding nodes and sets up proper binding. If, on the other hand, the two services to be bound are found in two separate worlds (e.g. a LONWorks device and another JINI enabled device), it handles binding with the original approach.

To test our design we were setting up a simulation environment consisting of some virtual JINI enabled (household) devices. The services described so far are self-contained entities implementing the abstract classes. When written as JINI services, they have to be split up into a lightweight proxy object, which is downloaded and executed by clients throughout the network, and a backend server, the service itself that does the actual work. Communication between proxy object and backend server is done using TCP/IP sockets. While each virtual JINI device has its own backend server, all LONWorks devices share a single *LONWorks Socket Server* instance. The LONWorks Socket Server is started only a single time when the JINI environment comes up. Therefore, it acts as backend server for all LONWorks proxy objects, keeping track of all states, but providing different ports for different devices.

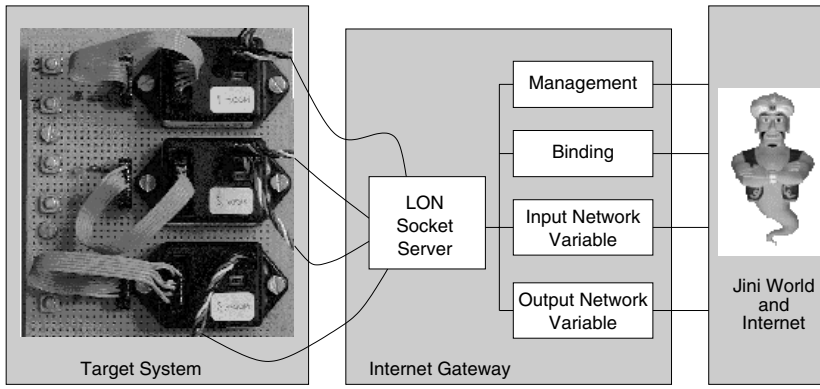


Fig. 6. LONWorks socket server

So to say, the task of the LONWorks Socket Server is to throw a bridge across LONWorks and the JINI world. Note, that it can be used for other Internet applications as well (Figure 6). The information and operations it provides depends on the requirements of the proxy object using it. For the first step, the simple gateway, it needs to offer write access to the values of the input variables and read access to the values of the output variables. For the integration of binding into JINI, it is able to create bindings between two variables, delete bindings between two variables and give information about already established bindings.

A simple type of binding queries at startup the Registrar for all available switching and switchable devices, makes them available for the user for selection, and waits until a source and a target device are chosen. From this time on, it observes the source – a switching device – if it changes its state, and, when this event happens, propagates the information to the target – a switchable device. Written in JAVA, this binding always requires:

- a message to be sent from the source device to the binding indicating that the switch has been pressed,
- process time from the hosting computer for deciding where to route this information to and
- a message being sent from the binding to the target device containing the information of the change-of-state.

While we cannot avoid this resource occupation during binding in the case of a JINI to JINI, JINI to LONWorks or LONWorks to JINI binding, it is possible for a more clever binding object to delegate this work to the LONWorks firmware in the case of a LONWorks to LONWorks binding. This reduces overhead and reaction time a lot. Figure 7 compares once more JINI and LONWorks binding of two LONWorks devices. For simplicity, the Internet LONWorks Socket Server layer has been omitted, and the proxy objects are allowed to talk directly to the LONWorks devices.

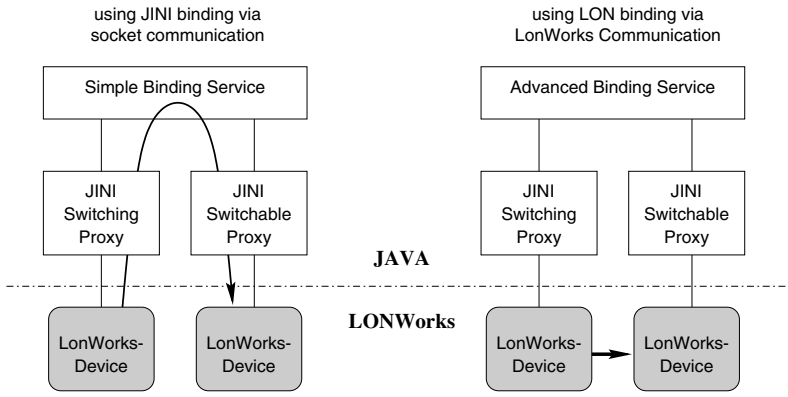


Fig. 7. Simple and advanced bindings

6 Conclusion

Today's users have increased needs for an easy way to connect sensors and actuators of the field area to upper networks and to control devices of lower networks from a remote location. Hence, this article examined the use of JINI as one way to easily connect devices of a fieldbus system to an upper network and to be able to remotely control them. As shown at the beginning of this article JINI is not the only solution for spontaneous networking. Currently, UPnP seems to provide a solid alternative to JINI and time will show, how vendors offer products for both systems. Therefore, our next step will be to compare a UPnP-enabled LONWorks-system (to be developed) with our JINI-enabled one.

References

1. T. Berners-Lee, L. Masinter, M. McCahill, *RFC 1738: Uniform Resource Locators*, 1994.
2. D Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, D. Winer, *Simple Object Access Protocol*, 2000.
3. T. Bray, J. Paoli, C. M. Sperberg-McQueen, *Extensible Markup Language*, 1998.
4. R. Droms, *RFC 1531: Dynamic Host Configuration Protocol*, 1993.
5. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *RFC 2616: Hypertext Transfer Protocol*, 1999.
6. W. Kastner, C. Krügel, H. Reiter: *Jini: ein guter Geist für die Gebäudesystemtechnik*, Proceedings JIT'99, Springer-Verlag, pp. 213–223, 1999.
7. D.C. Plummer, *RFC 0826: Ethernet Address Resolution Protocol*, 1982.
8. J. Postel, *RFC 0791: Internet Protocol*, 1981.
9. J. Postel, *RFC 0793: Transmission Control Protocol*, 1981.
10. The JINI Connection Technology, `http://www.sun.com/jini/`
11. The LONWorks Website, `http://www.lonworks.com/`
12. The Official Java Website, `http://java.sun.com/`
13. The Universal Plug and Play Forum, `http://www.upnp.org/`

Reusing Single-User Applications to Create Multi-user Internet Applications

Stephan Lukosch and Jörg Roth

Computer Science Department
University of Hagen
D-58084 Hagen
{Stephan.Lukosch,Joerg.Roth}@Fernuni-hagen.de

Abstract. Although there are many groupware platforms existing nowadays, collaborative multi-user applications are not yet widely accepted by end-users. In contrast to single-user applications, groupware applications often still have prototypical character and are lacking software quality. In this paper we introduce a three-step approach for reusing existing single-user applications for collaboration-aware multi-user applications. The three-step approach is based upon on our toolkit *DreamTeam* and its extension *DreamObjects*. By offering services for communication and coordination as well as data management and user interface development they significantly simplify the transformation of single-user applications into collaboration-aware applications. At the end of the paper we validate our approach with two examples: a diagram tool and a publicly available spreadsheet tool.

1 Introduction

Currently, the Internet offers a big set of applications, which allow groups or teams to collaboratively work on a joint task, e.g. Email, videoconferencing systems, newsgroups, or chat tools. Even though these applications cover a wide range of collaborative tasks, some specific activities cannot be handled conveniently with current Internet applications: to collaboratively edit a shared document in real-time (e.g. a text document or a spreadsheet) one usually has to develop a new tool, a difficult and time-consuming process: in addition to the actual application's task (e.g. editing texts or spreadsheets) network connections between collaborating users have to be supported, shared data have to be managed, and specific group functions have to be provided.

There exist two major approaches for developing collaborative applications [6]:

- *Collaboration-aware* applications are especially designed for collaborating teams. A collaboration-aware application usually has to be developed 'from-scratch' but offers a huge variety of group-specific services to end-users.
- *Collaboration-transparent* applications are single-user applications, which are run in a collaborative environment (e.g. a shared windows system).

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 79–90, 2001.

The latter approach saves development and usage costs, since single-user applications often provide high quality and users can use them without too much additional learning efforts in a multi-user environment as well. However, this approach induces two major problems: data management is hidden inside the application, thus data consistency can hardly be achieved; in addition, single-user applications, by definition, do not offer any group-specific services, and can hardly produce any group feeling (so-called *collaboration awareness*).

Our new approach combines the advantages of both approaches: we keep the quality, the functionality, and the user interface of an existing single-user application, but at the same time transform it into a truly collaboration-aware application. To minimise transformation costs, we offer a powerful runtime system, a set of programming abstractions for distributed data management, and a set of group-specific user interface elements.

Before describing the necessary transformation steps in detail, we discuss related work.

2 Related Work

DistEdit [4] and *DistView* [11] reuse existing single-user application as multi-user applications. DistEdit allows a transformation of editor programs. Although the transformation does not require much effort, it uses a floor control mechanism, which only allows one user at a time to edit a document and thus prevents real concurrent work.

DistView supports synchronous collaboration by distributing application windows. Each user can export one of his DistView windows to a central window server, from which another user may import the window. Interface and data objects are replicated to the importing site. Since all interface objects, e.g., a scrollbar or the window itself, are replicated, all users have the same view on the document. Concurrent work in large documents may lead to so called *scroll-wars*.

Besides DistEdit and DistView several collaborative toolkits have been developed during the last years.

GroupKit [12] is a package for implementing shared applications under Tcl/Tk. A library offers services for session management, communication, and shared dialogue management. A program library contains basic services for standard problems, covering session management, communication, and distributed user interfaces.

Dolphin [17] is a co-operative hypermedia system for co-operatively editing hypermedia documents. It is written in Smalltalk and provides a single hard-coded application, a shared hypermedia editor. The underlying platform *COAST* [16] offers general services for synchronous, document-based groupware.

Suite [2] extends a framework for developing single-user applications by mechanisms supporting groupware aspects. A Suite application consists of a module, which runs on a central server, and replicated dialogue managers. Because of their replication, dialogue managers are able to offer individual user interfaces for each user of a collaborative session.

Habanero [10] has fully been implemented in Java. It focuses on making Java applets available in a distributed environment. The applets must be available as source code and in most cases can be converted into a distributed applet (called *Hablet*). Interface events are distributed via Habanero's specific event distribution mechanism. If a specific applet does not fit with this distribution mechanism, it is quite difficult to transform it into a collaborative applet.

Each of these platforms significantly restricts the distribution architecture, the data model and the user interface of the application to be developed, and thus is not adequate for transforming existing single-user applications into pretentious collaboration-aware multi-user applications. In the following, we present our approach.

3 The Three-Step Approach

When transforming an existing single-user application into a collaboration-aware multi-user application, the transformation platform has to meet certain requirements: while preserving the functional core of the single-user application it must allow the developer ("transformer") to seamlessly integrate group-specific services. Though our concept does not depend on a specific object-oriented language, the *DreamTeam/DreamObjects* platform has been implemented in Java and thus can only be used for Java applications. While DreamTeam [13] provides a set of collaborative services, DreamObjects [8] focuses on shared data management.

The transformation consists of three steps:

1. Integrate the application into the platform's runtime system.
2. Organise shared data.
3. Add awareness services.

In the following, we describe these three steps in more detail.

3.1 Platform Integration

In order to collaborate, users must be able to plan and schedule sessions and to inform all group members about their plans in time. The DreamTeam runtime system offers a palette of services for setting up, coordinating, and scheduling sessions as well as for informing group members about planned and active sessions [14]. To provide these services, the runtime system must be able to control the application, i.e. the developer has to specify

- the application's name and icon,
- methods for starting and closing the application,
- the application's version, and
- optional methods for configuring the application.

The name and the icon of an application are used to identify the application in various overview and configuration windows. A user can, e.g., open a window,

which shows all active applications. While a single-user application usually provides a main method for starting the application and a menu entry for closing the application, the multi-user application is started and closed by the DreamTeam runtime environment via methods to be provided by the application. In a distributed system, different versions of the same application may run at different locations. To cope with potential inconsistencies, each application has to provide its actual version number as well as version numbers of compatible older versions. When the session manager detects conflicting versions, it prevents the application from starting and informs the corresponding users.

Before being started, an application may be configured; a teacher, e.g., may identify a set of slides before he starts a session based upon these slides.

3.2 Shared Data

Usually, a single-user application does not need to manage shared data, whereas data sharing and data consistency mean a major and difficult task for multi-user collaborative applications. To transform the data objects of a single-user application into shared ones, the following steps have to be performed:

1. Identify all data objects that have to be shared and define an adequate distribution mode for each such object.
2. Configure shared objects for consistency.
3. Couple the user interface with shared data objects.
4. Replace a shared object's constructor-based creation by its registration with the runtime environment.

In the following we describe these steps in more detail. If the architecture of a single-user application follows a style like *Arch* [18], *MVC* [5], or *PAC* [1] – which all postulate the a clear separation of data, functional core and user interface – its data objects can easily be identified.

The DreamObjects approach is based upon *substitutes* [8], a concept similar to the substitution principle in object-oriented languages. Substitutes offer the same interface as the substituted data object, and thus can easily be used as placeholders. The developer does not have to provide any additional classes, he uses the substitute like a local object, all distribution mechanisms are hidden in the substitute's methods.

Since discussions about the best distribution mode are still going on [15]. Our platform supports replicated as well as central data objects. While replicated objects may be used for highly responsive tasks, e.g. in group editors, central objects may be used for objects with extensive data but minor data exchange.

To define the distribution mode of a data object, it has to provide an additional interface. Just in case of a replicated object, the developer has to implement one additional method. Fig. 1 shows a class diagram for a replicated object class *SampleRObj* and a central object class *SampleCObj* in UML Syntax. The class *SampleRObj*, e.g., implements the interface *ReplicatedObject* and thus is replicated. The corresponding substitute class *SampleROSubstitute* can either be

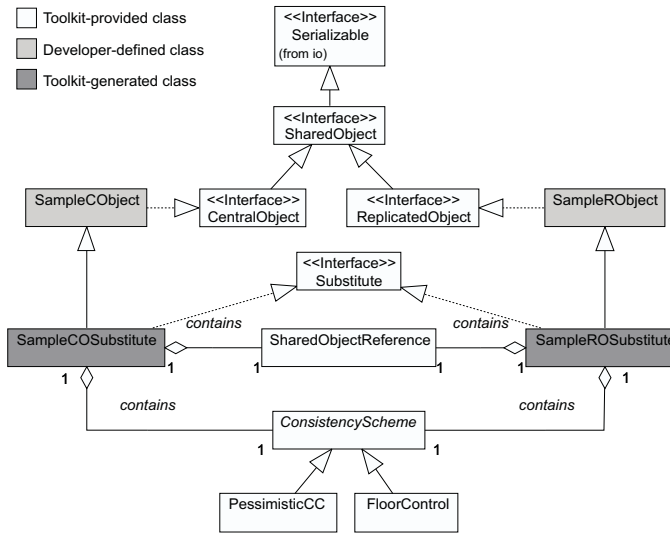


Fig. 1. Shared object class diagram

generated from the command line or the developer can leave it to the runtime environment to generate the corresponding substitute. Upon registration, the runtime environment uses these classes to replace the developer-defined classes. The substitute classes provide the *Substitute* interface, which offers necessary methods for the runtime environment. The aggregated *SharedObjectReference* is used to identify a shared object at runtime.

The aggregated *ConsistencyScheme* defines an object's consistency properties. In a multi-user scenario several users may manipulate an object simultaneously. To allow a maximum of concurrency, our platform provides a very flexible concurrency control service [7].

Usually, not all methods of a shared object modify all components of the object and thus some methods can be executed simultaneously. For each shared object, the developer can define sets of mutually *exclusive methods (EM)*, which form the *object's exclusive method set (OEM)*. To allow simultaneous method execution, whenever possible, our concurrency control scheme uses multiple locks, one for each mutually exclusive method set.

Imagine, e.g., a collaborative sketch editor, where a replicated object keeps the history of the sketch. Among other methods the history object offers the methods *changeLine* and *removeLine*. The following source code shows how the execution of these methods can be mutually excluded by adding a set of exclusive methods to the object's exclusive method set.

```
EM em=new EM("Draw.HistoryVector");
em.addMethod("changeLine");
em.addMethod("removeLine");
oem.addExclusiveMethodSet(em);
```

In contrast to single-user applications the data objects of a multi-user application may be manipulated unnoticed from the local application. Thus, there is a need for an application to react on remote changes. Our toolkit offers a flexible *object coupling service* [8], which allows the developer to trace changes in a shared data object and to propagate these changes to the user interface. This service is realised via a kind of extended callback mechanism, which avoids the confusing program code [9] of the normal callback mechanism. It allows to restrict the passed information to the needs of a developer-defined listener method. For this the developer has to define a *method mapping* between a shared object's method and a corresponding listener's method; the listener's method is called whenever the shared object's method is executed; its parameters can be composed from the shared object's method parameters, the method call result, and may contain information about the site, which called the method. Thus a method mapping consists of a shared object's method name, a listener's method name, and the listener's method parameters.

The following source code shows the method prototype of a method used to add a line to the sketch history:

```
public void addLine(int x1,int y1,int x2,int y2,int c);
```

The next source code example shows how these arguments are mapped to two different user interface methods:

```
CallListenerConfig config=
  new CallListenerConfig(CallListenerConfig.METHOD_MAPPING);
config.addMethodMapping(new MethodMapping("addLine","drawLine",
  new int[]{0,1,2,3})); // i.e. x1, y1, x2, and y2
config.addMethodMapping(new MethodMapping("addLine","setColour",
  new int[]{4})); // i.e. c
```

Finally, to use a shared object in the DreamObjects environment, the object's constructor-based creation has to be exchanged by a registration with the runtime environment, which enables the runtime environment to initialise the shared data object: the runtime environment creates an instance of the shared object's substitute class, adds this instance to its object registry, informs all other sites about the newly registered object, and returns the substitute. Depending on the shared object's distribution mode either a replica or a reference is distributed. To register a shared object, the developer has to call a special registration method and provide the shared object's class name, a unique registration name and the used consistency scheme as arguments. The following source code shows how the history object of the sketch editor is registered:

```
history=(HistoryVector)om.registerObject("Draw.HistoryVector",
                        "history",new PessimisticCC(oem));
```

3.3 Awareness

Awareness is an important requirement for multi-user applications. As other group members are not physically present, a collaborative application has to

provide some group feeling: so-called *awareness widgets* offer group specific services, e.g. an overview about other users' current activities. In contrast to the issues discussed in steps 1 and 2, awareness explicitly addresses end-users. Only if a multi-user application has appropriate support for group functions and awareness included into the user interface, an application will be accepted by end-users.

Our platform supports three kinds of awareness widgets:

1. Widgets, which are offered by the runtime system.
2. Widgets, which can be used as building blocks from the DreamTeam class library.
3. New widgets, which are created by an application developer.

Using the first kind of widgets does not cause any integration costs. DreamTeam, e.g., offers a list of all users of a collaborative group, who are currently online, which is called the *online list*, where each user is represented by a small picture. Users in this list are not necessarily working in a collaborative session, but they are ready for collaboration. Using this widget has an effect of "hang out in the hallway" [3]. A user can perceive other users, which are willing to collaborate, and thus may be challenged to spontaneously initiate a collaborative session.

Widgets of the second kind are *distributed mouse pointers*, which may, e.g., be used for discussing shared documents. DreamTeam allows to easily integrate distributed mouse pointers into an existing application. Usually, a user interface in Java is created by subclassing predefined Java classes such as Frame, Panel or Canvas. For distributed mouse pointers, a similar set of DreamTeam classes has to be subclassed. Each class offers services identical to the original class, but mouse distribution is automatically provided in the background. The application can control the behaviour of the mouse pointers via the DreamTeam API. It can switch on and off the mouse distribution to other users. Because too many pointers may confuse a user, remote pointers may be enabled and disabled. In addition, the application can define a string to be displayed below the mouse shape (normally the user name), as well as a pointer colour.

Another awareness widget is the so-called *tracking window* (see fig. 2). Tracking windows can be used to follow another one's work, if, e.g., a shared document may be scrolled independently by different users. The tracking window shows the current contents of the other user's window in a 1:3 scale, thus one user can follow the scrolling of another user.

Usually, an application developer integrates awareness widgets of the first two kinds into an application, by adding just a few lines of code. If a developer is not satisfied with the above awareness widgets, he can develop his own widgets. For this purpose, a widget can use information provided by the platform (e.g. the user list) and can register for group-related events (e.g. someone joins or leaves a session). These mechanisms help a developer to efficiently develop new awareness widgets, which can be collected in a class library and be reused in other applications.

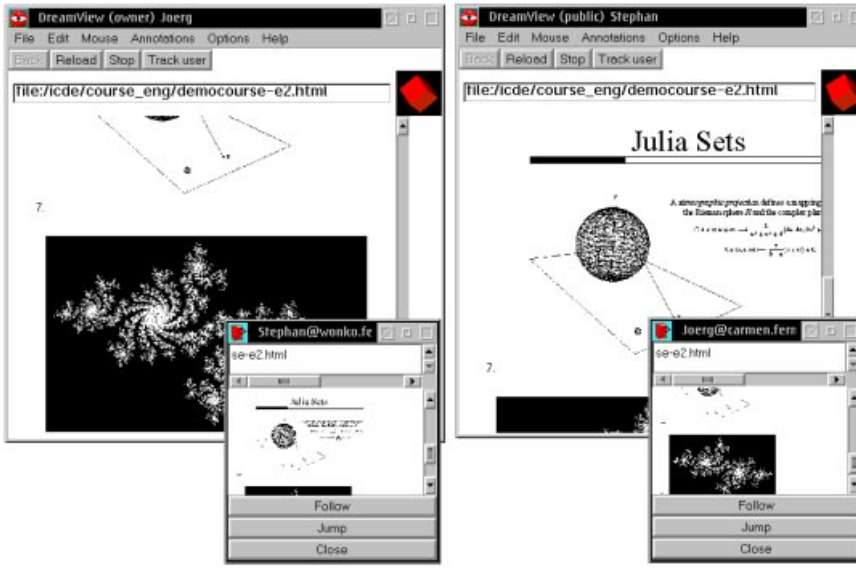


Fig. 2. Shared document browsing with tracking windows

4 Examples

We validated our approach by transforming some single-user applications. In the following we describe two example transformations.

4.1 A Diagram Editor

Our first example is a collaborative diagram editor (see fig. 3), which we derived from a single-user version.

This diagram editor allows the construction of diagrams such as flow charts or entity relationship diagrams. In contrast to a painting tool, diagram elements are not stored as bitmaps, and thus can easily be modified. According to our three-step approach the following transformation actions were performed:

Step 1: Embedding the application into the platform was simple: only one class, derived from the DreamTeam class library had to be coded. Since the superclass already contains some default methods, we only had to code some application-specific methods. The resulting Java class file contains less then fifty lines of code.

Step 2: The data class hierarchy of the single-user diagram editor consists of a container object class and a basic diagram element class, from which the different diagram elements, e.g. a rectangle or a circle, are derived. First, we transformed the container object class and the basic diagram element class into replicated objects. By defining different exclusive method sets, e.g. one for text operations and one for style changes, a maximum of concurrency was achieved. Next, the

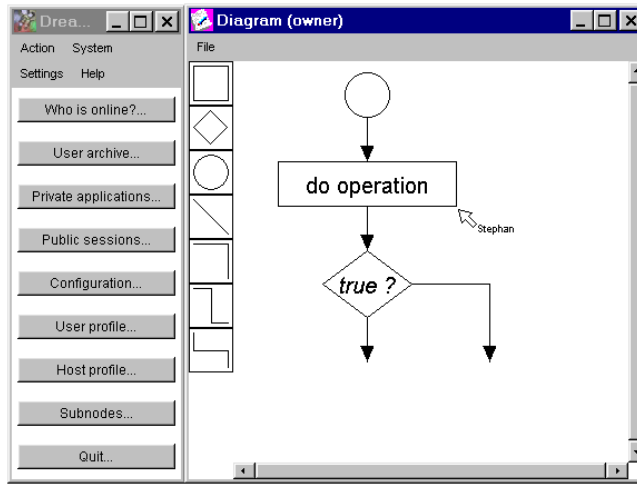


Fig. 3. A collaborative diagram editor

user interface was coupled to the container and the basic diagram element: whenever a diagram element is changed, added to, or removed from the container the user's view is updated.

Step 3: All built-in-widgets (e.g. participant windows) are available for the diagram editor. In addition, we integrated the distributed mouse pointer and the tracking window. The effort for integrating both elements was very small. The class, which does the painting inside the diagram had to be derived from a DreamTeam canvas class instead of the standard Java canvas. In addition, some lines of code were necessary to control these widgets, e.g. to enable or disable the pointers.

4.2 A Spreadsheet Tool

The next example is the single-user spreadsheet application, which is part of Sun's Java Development Kit JDK. The code for the tool is publicly available, but only supplied with small documentation, thus the transformation was a real challenge for our toolkit.

The steps to be performed were similar to the steps for the first application. In summary, the transformation could be done without considerable problems. In the following, we discuss our experiences.

4.3 Discussion

It is quite difficult to empirically assess the effort for transformations. Many factors influence the transformation process: on one hand, it is important how complex the application is and how well it was designed. On the other hand, the

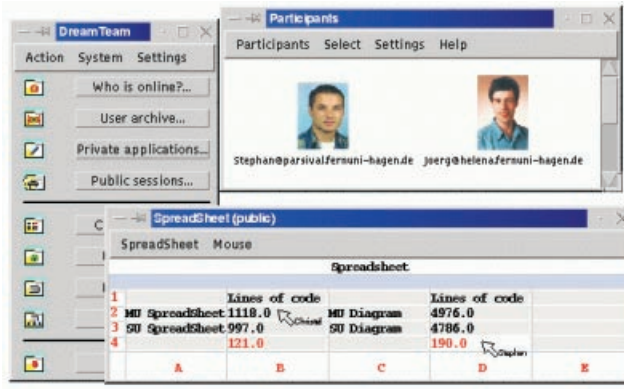


Fig. 4. A collaborative spreadsheet tool

Table 1. Actions to transform the applications

Single-user Diagram Editor (4786 lines of code)		
Step	**Actions**	**Lines of code**
1: Integration	Code a single class	43
2: Data	Transformation of the container class	15
	Transformation of the diagram element class	43
	Object coupling	14
3: Widgets	Derive the painting class and add statements to control widgets	75
	total	190
Single-user Spreadsheet tool (997 lines of code)		
Step	**Actions**	**Lines of code**
1: Integration	Code a single class	41
2: Data	Transformation of the cell class	18
	Object coupling	16
3: Widgets	Derive the painting class and add statements to control widgets	46
	total	121

transformation effort heavily depends on the skills of the transforming individual. Table 1 summarises the transformation efforts for both examples, measured in lines of code per transformation step.

The effort for reverse engineering the original application has not been included into the table. In total, one day of work was needed for each application.

5 Conclusion

Reusing existing single-user applications for collaborative scenarios is an important step for reducing development costs and increasing end-user acceptance. On the other hand, collaborative applications have to provide for collaboration awareness. Our approach minimises the effort for changing the single-user application, but at the same time adds collaboration awareness and group functions

to the collaborative application. Because of the simple transformation steps, in many cases even poorly designed single-user applications can easily be transformed into collaborative applications. In contrast to other platforms, our platform supports a variety of architectures, distribution mechanisms, concurrency schemes, and can thus be used for transforming a variety of single-user applications in an adequate way. We validated our approach with various single-user applications; two of them were discussed in this paper.

References

1. Gaëlle Calvary, Joëlle Coutaz, and Laurence Nigay. From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW. In *Human Factors in Computing Systems: CHI'97 Conference Proceedings*, pages 242–249. ACM, 1997.
2. Prasun Dewan and Rajiv Choudhary. A High-Level and Flexible Framework for Implementing Multiuser Interfaces. *ACM Transactions on Information Systems*, 10(4):345–380, October 1992.
3. H. Gajewska, M. Manasse, and D. Redell. Argohalls: Adding Support for Group Awareness to the Argo Telecollaboration System. In *Proceedings of the 8th annual ACM symposium on User interface software and technology*, pages 157–158, 1995.
4. Michael J. Knister and Atul Prakash. DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors. In *Proceedings of the ACM 1990 Conference on Computer Supported Cooperative Work*, pages 343–355, Los Angeles, California, USA, October 1990.
5. Glenn E. Krasner and Stephen T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, August 1988.
6. J. C. Lauwers and K. A. Lantz. Collaboration awareness in support of collaboration transparency: requirements for the next generation of shared window systems. In *CHI '90 Conference on Human factors in computing systems, special issue of the SIGCHI Bulletin*, pages 303–311, Seattle, Washington, USA, April 1990.
7. Stephan Lukosch and Claus Unger. Flexible Synchronization of Shared Groupware Objects. *ACM SIGGROUP Bulletin*, 20(3):14–17, December 1999.
8. Stephan Lukosch and Claus Unger. Flexible Management of Shared Groupware Objects. In *Proceedings of the Second International Network Conference (INC 2000)*, pages 209–219, University of Plymouth, Great Britain, July 2000.
9. Brad A. Myers. Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs. In *Proceedings of the 4th annual ACM symposium on User interface software and technology*, pages 211–220, Hilton Head, South Carolina, USA, November 1991. ACM SIGGRAPH.
10. NCSA Habanero Homepage. http://havefun.ncsa.uiuc.edu/habanero.
11. Atul Prakash and Hyong Sop Shim. DistView: Support for Building Efficient Collaborative Applications using Replicated Objects. In *Proceedings of the ACM 1994 Conference on Computer Supported Cooperative Work*, pages 153–164, Chapel Hill, NC, USA, 1994.
12. Mark Roseman and Saul Greenberg. Building Real-Time Groupware with GroupKit, A Groupware Toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1):66–106, March 1996.

13. Jörg Roth. 'DreamTeam': A Platform for Synchronous Collaborative Applications. *AI & Society*, 14(1):98–119, March 2000.
14. Jörg Roth and Claus Unger. Group Rendezvous in a Synchronous, Collaborative Environment. In *11. ITG/VDE Fachtagung, Kommunikation in Verteilten Systemen (KiVS'99)*, March 1999.
15. Jörg Roth and Claus Unger. An extensible classification model for distribution architectures of synchronous groupware. In *Proceedings of the Fourth International Conference on the Design of Cooperative Systems (COOP2000)*, Sophia Antipolis, France, May 2000. IOS Press.
16. Christian Schuckmann, Lutz Kirchner, Jan Schümmer, and Jörg M. Haake. Designing object-oriented synchronous groupware with COAST. In *Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work*, pages 30–38, Cambridge, MA, USA, July 1996.
17. N. A. Streitz, J. Geißler, J.M. Haake, and J. Hol. DOLPHIN: Integrated Meeting Support across LiveBoards, Local and Remote Desktop Environments. In *Proceedings of the ACM 1994 Conference on Computer Supported Cooperative Work*, pages 345–358, Chapel Hill, NC, USA, 1994.
18. The UIMS Tool Developers Workshop. A Metamodel for the Runtime Architecture of an Interactive System. *ACM SIGCHI Bulletin*, 24(1):32–37, January 1992.

Graph-Theoretic Web Algorithms: An Overview

Narsingh Deo and Pankaj Gupta

School of Computer Science, University of Central Florida, Orlando, FL 32816, USA
{deo,pgupta}@cs.ucf.edu

Abstract. The World Wide Web is growing rapidly and revolutionizing the means of information access. It can be modeled as a directed graph in which a node represents a Web page and an edge represents a hyperlink. Currently the number of nodes in this gigantic Web graph is over four billion and is growing by more than seven million nodes a day—without any centralized control. The study of this graph is essential for designing efficient algorithms for crawling, searching, and ranking Web resources. Knowledge of the structure of the Web graph can be also exploited for attaining efficiency and comprehensiveness in Web navigation. This paper describes algorithms for graph-theoretic analysis of the Web.

1 Introduction

The World Wide Web (WWW or Web) has revolutionized the way we access information. As of March 2001, the Web is estimated to have over four billion pages, more than 28 billion hyperlinks, and is growing rapidly at the rate of 7.3 million pages a day (Moore and Murray [15]). The Web can be viewed in two very divergent ways. From an internal standpoint, it is a distributed TCP-compliant application. We are interested in the other equally-important way of viewing the WWW, which is external and extensional. From this standpoint, the WWW is a vast and continuously growing repository of information: textual as well as audio and video. The sheer mass of material and the apparently chaotic nature of the WWW can make locating, acquiring, and organizing information both time consuming and difficult. This runs exactly opposite to the promise of the WWW, namely that all information can be made readily available to the world's population all of the time. The first challenge to overcome in attaining this laudable goal would be to find methods for efficiently and comprehensively navigating the WWW. In WWW-intrinsic terms, efficiency means that one wants to minimize the number of links that must be followed to reach a desired piece of information from a designated starting point. Comprehensiveness means that the most relevant information is actually obtained during the navigation. At present, neither efficiency nor comprehensiveness can be reliably achieved.

Despite its chaotic appearance, the WWW is highly structured, but in a statistical sense. Models have been proposed which reproduce certain experimentally determined features of the WWW and these features could be exploited to attain efficiency and comprehensiveness in the WWW navigation. In this paper, we present graph-theoretic models and algorithms for the Web. The paper is organized as follows. Section 2 surveys the random-graph models that explain the Web structure.

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp 91–102, 2001.

Section 3 describes algorithms for efficient search and identification of Web communities. Section 4 presents the concluding remarks. The description of the graph-theoretic terms used in this paper can be found in a standard graph-theory book such as [7].

2 Random-Graph Models of the Web

The Internet is a constellation of data-communications technologies. Internet topology and traffic are defined by data-communications lines and resource allocation issues in entities like routers or operating systems. However, the Web topology is an aggregate sum of individual decisions and is not influenced by data communications or systems considerations. Web links reflect semantically motivated, intentional acts by human beings. The Web is different from a distributed database in the sense that there is no uniform structure, integrity constraints, transactions, standard query language, or a data model. The uncontrolled, decentralized, and rapid growth of the Web is absent in a distributed database. Moreover, the key features of a database such as reliability, recovery, *etc*., are not present in the Web. Recent studies show that the Web structure resembles collective behavior reminiscent of complex physical systems and thermodynamics. The Web can be modeled as a directed graph, where each node represents a page and each edge, a hyperlink.

Empirical study is a useful technique for understanding the Web structure. Pirolli *et al.* [18] performed the earliest study of link structure of the Web. They studied three kinds of graphs to represent the strength of association among Web pages: hypertext-link topology, inter-page text similarity, and flows of users through a locality. Kumar *et al.* [14] found that despite the chaotic nature of content creation on the Web, there exist well-defined communities. Barabási and Albert [3] analyzed the induced graph of 325,729-node *www.nd.edu* and extrapolated the expected distance between any two nodes in the Web graph (of 2.1 billion nodes) to be about 19. They also found the inverse power-law degree distribution of a Web page. Kleinberg *et al.* [13] report the distribution of bipartite cores on the Web from the result of *Alexa* crawl. In one of the most extensive empirical studies, Broder *et al.* [5] analyzed the link structure of 203 million pages and 1.5 billion links. Their study found that the Web graph has four distinct regions. There exists a giant strongly connected component (SCC). There is a set of newly formed nodes called IN having only outgoing links and another set of nodes called OUT having only incoming links (like corporate and e-commerce sites). There exists a directed path from each node in IN to SCC, and a directed path from SCC to each node in OUT. There exists a directed path from nodes in IN to a set of nodes called TENDRILS and from a separate set of TENDRILS to nodes in OUT. Bar-Yossef *et al.*[2] studied random walks on the Web for uniform sampling. In their implementation of random walk called *WebWalker*, a *d*-regular, undirected graph was built using the resources such as HTML text analysis, search engines, and the random walk itself.

In this section, we present various models of random graph that have been proposed to describe the growth of the Web.

2.1 Erdös-Rényi Model

The earliest model of a random graph was proposed by Erdös and Rényi [9] (example in [8], p. 8) in which we start with n isolated nodes, and each of the $\frac{n(n-1)}{2}$ pairs of nodes is connected by an edge with a uniform probability p. This static model does not represent the real-life WWW because it does not take into account:

(i) The increasing number of nodes in the Web and
(ii) The non-uniform probability of edge formation between node pairs.

2.2 Small-World Models

In 1967 Milgram ([20], p. 23) articulated the basic properties of small-world networks, based on the well-founded folklore that each individual is indirectly linked through a short chain of acquaintances to practically anyone else in the world. Sparseness, small diameter, and cliquishness are the three properties that characterize small-world networks. Erdös-Rényi type random graphs have a small diameter, but they lack the cliquishness present in the small-world networks. The small-world effect has been identified in disparate contexts including neural network of the worm *C. elegans*, epidemiology, power-grid networks, collaboration graph of film actors, and the WWW [20]. Two variations of small-world model have been proposed. They are the edge-reassigning and edge-addition small-world network.

Edge-Reassigning Small-World Network
Watts and Strogatz [20] proposed the edge-reassigning small-world model. In this model, evolution starts with a ring lattice having each node connected to its d nearest neighbors. Each of the $\frac{n \cdot d}{2}$ edges is randomly removed and reassigned to distant nodes with a probability σ in a round robin fashion (example in [8], p. 10). This model has two properties: characteristic-path length that measures the separation between two nodes (global property), and clustering coefficient that measures the cliquishness of neighborhood of a node (local property). The chief feature of this model is that a region of values for σ produces evolution, unlike the Erdös-Rényi type random-graph evolution. When $\sigma = 0$, the original regular-graph remains unchanged. As σ increases from 0 to 1, the characteristic path-length decreases rapidly. As edges are reassigned from a clustered neighborhood, a poorly clustered, small-world random network is formed. When $\sigma = 1$, we get an Erdös-Rényi type random graph.

Edge-Addition Small-World Network
The edge-addition small-world model was proposed by Newman *et al.* [16]. In this model, $\frac{p \cdot d \cdot n}{2}$ new edges are added randomly to an existing ring lattice (example in [8], p. 11). Here, p, d, and n denote the new-edge probability, degree of each node in the original ring-lattice, and the number of nodes, respectively.

Newman *et al.* [16] considered the following situation in a small-world network. Some fraction f of nodes is populated by individuals who will contract a disease if exposed to it. The probability $P(j)$ that a randomly chosen node i belongs to a connected cluster of j nodes is determined. If any node i contains an infected individual, $P(j)$ is the probability that j people will be consequently infected because there are very short paths from the original infected individual to all others in the cluster. Newman *et al.* derive the threshold value of f at which the expected size j of infectious outbreak diverges.

The spread of epidemics in a population bears an interesting analogy to the spread of viruses on the WWW. The small-world characteristic of the Web aids in the spread of a virus epidemic. A file meant for general distribution on a popular site, if infected by a virus, can become the source of a major outbreak. The Web pages of a popular Web site are vulnerable, because these pages are part of a cluster. Once infected, they propagate the virus through their immediate neighbors and cause a global epidemic thereby causing enormous loss of time and money. The study of small-world network is useful in predicting the nature of a virus outbreak and its effect on the Web.

2.3 The Preferential-Attachment Model

The small-world models cannot be applied directly to the WWW because the number of pages in the Web is not fixed. These models do not accommodate a birth/death process in which new pages are created, how new links are formed (possibly through editing old pages), and how both links and pages can be deleted. Two new models have been proposed recently to explain some of the empirical findings concerning the overall WWW structure, taking on board the reality of the birth/death process.

The preferential-attachment model [3] starts with a small, null graph having a finite number of nodes (n_0). At each successive time step, a new node with a bounded number of outgoing edges is added to the network. The probability that a link from the new node goes to an existing node i is $\frac{d_i}{\sum_j d_j}$, where the denominator sum runs over all existing nodes (example in [8], p. 14). Thus, a newly-introduced node is more likely to be adjacent to a node with high degree.

The preferential-attachment model reproduces one of the most significant empirical findings about the WWW, namely, the probability that a page or a node i has degree d_i is $\frac{A}{(d_i)^c}$, where A is proportional to the square of average degree of the network and c is a constant. The exponent c was empirically found to be about 2.9, and it is independent of the number of edges being added at each time step.

The preferential-attachment model does not allow for the reconnection of existing edges. Also, addition of new edges takes place only when new nodes are added in the system. However, in real-life, new links are added continuously between old nodes as well.

2.4 The Web-Site Growth Model

Huberman and Adamic [11] have proposed another model that exhibits an inverse, scale-free power law for the probability that a Web site s has N_s pages. The term Web site in their study is defined as a registered domain-name on the Internet. In the Web-site growth model, the number of pages added to a site s at any given time is considered proportional to those already existing on the site. The equation has the form

$$N_s(t+1) = N_s(t) + g(t+1) \cdot N_s(t),$$

where, $N_s(t)$ = the number of pages at site s at time step t, and
$g(t)$ = universal growth rate, which is independent of a site.

Due to the unpredictable nature of growth of a site, $g(t)$ fluctuates about a positive mean g_0, and it can be expressed as

$$g(t) = g_0 + \xi(t),$$

where, g_0 = the basic, constant growth rate, and $\xi(t)$ = a Brownian motion variable.

The expected value of $N_s(t)$ is

$$N_s(t) = N_s(0) \cdot e^{(g_0 \cdot t + \upsilon_t)},$$

where, υ_t is a Wiener process such that $\upsilon_t^2 = e^{(\mathrm{var}(g) \cdot t)}$ and $var(g)$ is variance of growth rate $g(t)$ of the Web site.

The probability of $N_s(t)$ pages at a site s is given by a weighted integral, which eliminates dependence on time t and yields a scale-free inverse power-law $\frac{c}{(N_s)^{\gamma}}$, where c is a constant and exponent γ is in the range $[1, \infty]$.

In a short note, Adamic and Huberman [1] criticized the preferential-attachment model for its prediction that older pages have larger in-degree than newer pages. Empirical data appears to show no correlation between the age and in-degree of a Web page. They argue that the growth rate of degree of a page depends on the current degree of page and not on its age.

3 Search and Page-Evaluation Algorithms

The enormous size and rapid growth of the Web makes it difficult for an individual to locate information and navigate by just employing Web addresses. A search engine is useful for locating information in the vast space of the Web. Ranking the pages returned by a search engine is done by ranking-function heuristics based on the frequency of occurrence of keywords, and sometimes on the position of keywords in the page. However, such strategies may not deliver correct information. For example, some Web pages may have a keyword repeated many times to attract Web traffic or gain favorable ranking.

One of the factors in the effectiveness of a text-based search engine is the number of Web pages indexed in its database. As of March 2001, *Google* had the largest database with 1,346 million pages indexed. *Fast* had 575 million pages, *Inktomi* had 500 million pages, and *Altavista* had 350 million pages indexed. The search engine

with the largest index (*Google*) covers fewer than 34% of the present Web pages and this disparity is going to increase in future. Vast inconsistency in the number of hits for two similar queries on a search engine is another serious problem, *e.g.*, a query for "Oscar award" on *Altavista* search engine resulted in 1,480 hits while "award Oscar" yielded 1,208,710 hits. More examples of inefficient search results appear in [8].

A higher level of abstraction above a Web page is helpful in understanding the Web topology. In this section, we explore how this abstraction can be used as a tool to identify order and hierarchy in the Web. This understanding is crucial for developing novel search algorithms and enhancing the present search-engine technology.

3.1 The WWW Communities

A WWW site has conventionally meant a collection of pages defined by design. In some cases, a site is coherent with respect to some semantical interpretation, *i.e.*, the pages focus on different aspects of one topic, but more often the home page is just the hub of the collection. A WWW site can be also defined through link counting. Here, we consider some notion of locality, *e.g.*, some subnet IP address range and then measure the ratio of the links among all pages inside the range to all the links going outside the range. A more precise formulation can be obtained by computing SCCs. Gibson *et al.* [10] identified two kinds of pages that together make possible a computational concept of a WWW community of pages. One kind of pages represent authorities focusing on a topic, while the other kind represent hubs which point to many authorities.

The abstract community of hubs and authorities arise due to the abundance of Web pages that can be returned for any broad-based query. The goal of a search method is to return a small set of "authoritative" pages for the query. The hyperlinks can be exploited for understanding the inherent Web communities. A link from a page u to a page v can be viewed as conferral of authority on page v. However, there are many links created which have no meaning with respect to conferral of authority. Counting the incoming links of any Web page is not the complete solution to the problem of identifying authority.

3.2 HITS Algorithm

Kleinberg [13] proposed an iterative algorithm, called HITS (Hypertext Induced Topic Search), for identifying authorities and hubs using the adjacency matrix of a subgraph of the WWW. For a broad-topic search, the algorithm starts with a root set S of pages returned by a text-based search engine. The set S induces a small subgraph focused on the query topic. This induced subgraph is then expanded to include all the nodes that are successors of each node in set S. In addition, a fixed number of predecessors of each node in the set S are also included. Let G be the graph induced by the nodes in this expanded node set. It should be noted that the links that are used purely for navigation within a Web site are not included in this graph G. For each node, x, in the graph G a non-negative authority weight $a(x)$ and a non-negative hub weight $h(x)$ are computed. The authority and hub weights of all the nodes may be expressed as vectors $a()$ and $h()$, respectively. The elements of vectors $a()$ and $h()$ are initialized to one. In each iteration, $a(x)$ is replaced by the sum of $h(x_i)$'s of all the

predecessors of node x, and $h(x)$ is replaced by the sum of the $a(x_j)$'s of all the successors of node x. The iterations may be expressed as

$$a(x) = \sum_{v \to x} h(v), \quad \text{and} \quad h(x) = \sum_{x \to w} a(w).$$

The authority and hub scores are normalized in each iteration so that $\sum (a(x))^2 = 1$, and $\sum (h(x))^2 = 1$. This iterative process converges to yield the authority and hub vector for the initial query. If M is the adjacency matrix of the graph G, then the iterations of vector a when normalized, converge to the principal eigenvector of M^TM. Similarly, multiple iterations of the normalized vector h converge to the principal eigenvector of MM^T. Thus, HITS applies a link-based computation for identifying the hubs and authorities on a query topic.

3.3 Web-Page Evaluation

A common problem with search engines, that evaluate Web page relevance based on frequency of keywords, is that they can be biased by deliberate inflation of keywords in the Web pages. Another problem is that many Web pages do not contain the keywords that best describe what the Web page is known for or the services it provides. Evaluating the importance of a Web page, using the graph structure of the Web, can solve both these problems. We now present three algorithms for measuring the importance of a Web page.

PageRank Algorithm

Conventional search-engines have relied on matching keywords and strings in the Web pages to index and search the Web for information. *Google* uses the graph structure of the Web to produce better search results. It uses PageRank algorithm [17] that attempts to give a ranking to a Web page, regardless of its content, based solely on its location in the Web graph.

Here, i = a node (Web page) in the Web graph,
d_i^+ = out-degree of node i,
$w_1, w_2, \ldots, w_k$ = predecessors of node i,
η = normalization constant ($\eta < 1$), and
$PR(u)$ = PageRank of a Web page i.

The PageRank of a page i is given as

$$PR(i) = (1 - \eta) + \eta \cdot \left(\frac{PR(w_1)}{d_1^+} + \frac{PR(w_2)}{d_2^+} + \cdots + \frac{PR(w_k)}{d_k^+}\right).$$

The PageRank algorithm starts with assigning a rank of one to all pages and recursively computes the PageRank value for each page. The rank of a page is divided equally among its outgoing links. A page has high PageRank if pages having high PageRank point to it. The PageRank vector $PR()$ corresponds to the principal eigenvector of normalized adjacency-matrix of the Web graph. Normalization constant, η, is the probability that a random surfer does not follow an outgoing link of a page i and selects another page randomly.

Page Reputation

Computing reputation ranks relies on the model of a random "Web surfer", who is browsing the Web for pages relating to a certain topic τ. At each step, the surfer either jumps to a random page that contains the term τ, or follows a random outgoing link from the current page. As this process continues, a reputation value is computed, equal to the number of visits that the random surfer makes to a particular page.

One-Level Reputation Rank Algorithm

The reputation rank algorithm proposed by Rafiei and Mendelzon [19] converges to produce reputation ranks for each page w and term τ.

The reputation of a page w on topic τ is defined as the probability that a random surfer looking for τ visits page w. The notations used in the reputation algorithm are:

N_τ = total number of pages on the Web containing the term τ,

d_x^+ = number of outgoing hyperlinks from a page x,

ρ = probability that a random surfer selects a page uniformly at random from a set of pages containing the term τ,

$(1 - \rho)$ = probability that a random surfer follows an outgoing link from the current page, and

R = a matrix where a row corresponds to a Web page and a column corresponds to each term that appear in the Web page. Each element of the matrix, $R(w, \tau)$, is the reputation value of the Web page w with respect to term τ.

```
For every page w and term τ
     If τ appears in page w
          R(w, τ) = 1/N_τ
     Else R(w, τ) = 0
while R has not converged
    set R'(w, τ) = 0 for every page w and term τ
    For each link x→w
         R'(w, τ) = R'(w, τ) + R(x, τ) / d_x^+
    For every page w and term τ
         R(w, τ) = (1 - ρ) * R'(x, τ)
    If term τ appears in page w
         R(w, τ) = R(w, τ) + ρ/N_τ
```

The probability that the random surfer visits a page w at each step in a random jump is ρ/N_τ, if the page w contains term τ and is zero otherwise. If a page x is a predecessor of page w, then the probability that the surfer visits page w at k steps after

visiting page x is $(\frac{(1 - \rho)}{d_x^+})R^{(k-1)}(x, \tau)$, where, $R^{(k-1)}(x, \tau)$ is the probability that the surfer visits page x at step $(k - 1)$. The reputation algorithm calculates the probabilities iteratively.

Markov-Chain-Based Rank Method

Zhang and Dong [21] have proposed another ranking algorithm based on the Markov-chain model. The rank function of a page is defined as rank: S x Q, where S represents a set of Web pages and Q represents a set of user queries. The set of Web pages returned by a search engine, S, induce a Web subgraph $G = (V, E)$. This algorithm takes into account four parameters: relevance, authority, integrativity, and novelty. The similarity between the contents of a Web page and the user's query q is measured by the relevance (ω). The authority of a Web page (μ) is a measure of references made to the Web page. Integrativity (θ) is a measure of references made in the Web page. The novelty metric (ε) measures how a Web page is different from other pages.

While surfing, a user jumps from one page to another, and this process can be modeled as a homogeneous Markov chain. For a query q, $S = \{S_1, S_2, \cdots, S_n\}$ denotes the set of related Web pages found by the search engine. S can be viewed as the state space, where a Web page corresponds to a state. For a user surfing the Web at a time t, $p_i(t)$ is the probability that the user is browsing page S_i and p_{ij} is the probability that the user jumps to another Web page S_j by following an out-going link from page S_i.

Consider a random surfer viewing a Web page S_i at time t. Then at time $(t + 1)$, the random surfer has four choices: continue viewing Web page S_i, click on a link on Web page S_i and reach another page, use the "Back" option of browser to return to the previous page, or select another Web page from the results (S) of the search engine. The tendency matrix takes into account all these four choices available to the user by using relevance (ω), authority (μ), integrativity (θ), and novelty (ε). The tendency matrix W, derived from the graph G, is represented as

$$W_{ij} = \begin{cases} \omega \cdot sim(S_i, q), & \text{if } i = j \\ \mu, & \text{if } (v_i, v_j) \in E \\ \theta, & \text{if } (v_j, v_i) \in E \\ \varepsilon, & \text{Otherwise.} \end{cases}$$

Here, $sim(S_i, q)$ is the relevance of result S_i to the query q, $0 < \omega, \mu, \theta, \varepsilon < 1$, and $\omega + \mu + \theta + \varepsilon = 1$. Normalizing the tendency matrix W results in the transition probability matrix T for the set of Web pages S.

Therefore, $T_{ij} = \frac{W_{ij}}{\sum_{j=1}^{n} W_{ij}}$, and $T = (T_{ij})_{n \times n}$.

A homogeneous Markov chain's behavior can be determined by its initial distribution vector $T(0)$ and its transition probability matrix T, $T(t) = T(0)\ T^t$. A holomorphic and homogeneous Markov chain, $\{x_t, t \geq 0\}$, with state space S, transition probability matrix T, and initial distribution vector $T(0)$, converges to a

unique distribution, *i.e.*, $\lim_{t\to\infty} T(t) = D$. The rank of the Web pages is the ultimate distribution vector $D = \{\pi_1, \pi_2, \cdots, \pi_n\}$, which is the unique solution of the equation $DT = D$ that satisfies $\pi_i > 0$, $\sum_{i=1}^{n} \pi_i = 1$.

3.4 Small-World Algorithmics

Kleinberg [12] proposed an algorithm for finding the shortest or near-shortest path from one node to another node in a graph with small-expected diameter. The algorithm considers a two-dimensional grid with directed edges. The probability of an edge between nodes u and v is proportional to $[L(u, v)]^{-r}$, $(r \geq 0)$, where $L(u, v)$ is the distance between nodes u and v. Using an extension of the Watts-Strogatz model, Kleinberg devised a decentralized algorithm that finds a near-shortest path in expected time, polylogarithmic in the size of the graph. The algorithm considers the problem of passing a message from a node u to another node v. In each step, an intermediate node i chooses a contact node that is as close to the target node v as possible. It assumes that every node knows the location of the target node in the network as well as the location and long-range contact of all nodes that have met the message. Kleinberg proved that at $r = 2$, the decentralized algorithm takes best advantage of the geographic structure of the network and generates paths of length $O(\log n)$, where n is the number of nodes in the grid.

3.5 Related-URL and Topic-Distillation Algorithms

The graph topology of the Web can be exploited to discover novel search-techniques. Dean and Henzinger [6] proposed a search algorithm for finding Web pages related to a URL. A related Web page is one that addresses the same topic as the original page, but is semantically different. The Dean-Henzinger algorithm has following steps:

1. Build a vicinity graph for a given URL, *i.e.*, node U.

   The vicinity graph is an induced, edge-weighted digraph that includes the URL node U, up to B randomly selected predecessor nodes of U, and for each predecessor node up to B_F successor nodes different from U. In addition, the graph includes F successor nodes of U, and for each successor node up to F_B of its predecessor nodes different from the node U. There is an edge in the vicinity graph if a hyperlink exists from a node v to node w, provided both the nodes v and w do not belong to the same Web site.
2. Eliminate duplicate and near-duplicate nodes.

   Duplicate nodes are mirror sites or different aliases of the same Web page. Two nodes are defined as near-duplicate nodes if they have more than 95% of the links in common and each node has more than 10 links. The near-duplicate nodes are replaced by a node with links that are union of links of all the near-duplicate nodes.
3. Compute edge weights based on connections between Web sites.

   An edge between nodes on same Web site is assigned a weight 0. If there are m_1 edges directed from a set of nodes on one Web site to a single node on another Web site, then each edge is given an authority weight $1/m_1$. If there are m_2 edges directed

from a single node on one Web site to a set of nodes on another Web site, each edge is assigned a hub weight $1/m_2$. This prevents the influence of a single Web site on the computation.

4. Compute a hub and an authority score for each node in the graph.

The ten top-ranked authority nodes are returned as the pages that are most related to the start page U (modified version of HITS algorithm, Section 3.2).

Bharat and Henzinger [4] proposed a search-engine enhancement algorithm, based on the Web topology, called *Topic distillation*. *Topic distillation* is defined as the process of finding quality Web pages related to a query topic. The topic distillation algorithm solves three problems associated with the HITS algorithm. The first problem is the mutually reinforced relationship between Web sites where hub and authority score of nodes on each Web site increase. The other two problems are automatically generated links and presence of non-relevant nodes. If there are m_1 edges directed from a set of nodes on one Web site to a single node on another Web site, then each edge is assigned an authority weight (*edge_auth_wt*) of $1/m_1$. Similarly, if there are m_2 edges directed from a single node on one Web site to a set of nodes on another Web site, then each edge is assigned a hub weight (*edge_hub_wt*) of $1/m_2$. In addition, isolated nodes are eliminated from the graph. The hub weight and authority weight of each node is calculated iteratively as:

$\forall u \in V,$

$$a(u) = \sum_{(v,u)\in E} h(v) \times edge\_auth\_wt(v, u) \quad \text{and} \quad h(u) = \sum_{(u,v)\in E} a(v) \times edge\_hub\_wt(u,v).$$

The similarity between the query and the node returned by a search engine is defined as the relevance weight of the node. The relevance weight of each node is computed and the nodes whose relevance weights fall below a threshold level are eliminated.

4 Conclusion

The topology of the World Wide Web exhibits the characteristics of a new type of random graph, which at present, is only dimly understood. Recent studies have uncovered only a few fundamental properties of the Web graph. We believe there are still more subtle, but important graph-theoretic properties yet to be discovered about the Web. The rapid growth of the Web poses a challenge to the present search-engine technology. The solution for improving search quality involves more than just scaling the size of the search-engine index database. Graph-theoretic algorithms, that take into account the link structure of the Web graph, will lead to the development of better search engines and smart agents for providing relevant information to the end user, with efficiency and comprehensiveness.

References

1. Adamic, L., Huberman, B.: Technical comment to 'Emergence of scaling in random networks'. Science, Vol. 287. (Mar. 2000) 2115

2. Bar-Yossef, Z., Berg, A., Chien, S., Fakcharoenphol, J., Weitz, D.: Approximating aggregate queries about Web pages via random walks. Proceedings of the 26th Int. Conf. on Very Large Databases, Cairo, Egypt (Sept. 10-14, 2000) 535-544
3. Barabási, A., Albert, R.: Emergence of scaling in random networks. Science, Vol. 286. (Oct. 1999) 509-512
4. Bharat, K., Henzinger, M.: Improved algorithms for topic distillation in a hyperlinked environment. Proceedings of the 21st ACM SIGIR Conf. on Research and Development in Information Retrieval, Melbourne, Australia (Aug. 24-28, 1998) 104-111
5. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph structure in the Web: Experiments and models. Proceedings of the 9th Int. WWW Conf., Amsterdam, The Netherlands (May 15-19, 2000)
6. Dean, J., Henzinger, M.: Finding related pages in the World Wide Web. Proceedings of the 8th Int. WWW Conf., Toronto, Canada (May 11-14, 1999)
7. Deo, N.: Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall, Englewood Cliffs, NJ (1974)
8. Deo, N, Gupta, P.: World Wide Web: A graph-theoretic perspective. Technical Report CS-TR-01-001, School of Computer Science, University of Central Florida, Orlando, FL (Mar. 2001) (http://www.cs.ucf.edu/~pgupta/publication.html)
9. Erdös, P., Rényi, A.: On the evolution of random graphs. Publications of the Mathematical Institute of the Hungarian Academy of Sciences, Vol. 5. (1960) 17-61
10. Gibson, D., Kleinberg, J., Raghavan, P.: Structural analysis of the World Wide Web. WWW Consortium Web Characterization Workshop, Cambridge, MA (Nov. 5, 1998)
11. Huberman, B., Adamic, L.: Growth dynamics of the World-Wide Web. Nature, Vol. 401. (Sept. 1999) 131
12. Kleinberg, J.: The small-world phenomenon: an algorithmic perspective. Technical Report 99-1776, Department of Computer Science, Cornell University, Ithaca, NY (Oct. 1999)
13. Kleinberg, J., Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: The Web as a graph: Measurements, models and methods, Lecture Notes in Computer Science, Vol. 1627. Springer-Verlag, Berlin Heidelberg New York (July 1999) 1-17
14. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Trawling the Web for emerging cyber-communities. Proceedings of the 8th Int. WWW Conf., Toronto, Canada (May 11-14, 1999)
15. Moore, A., Murray, B. H.: Sizing the Web. Cyveilliance, Inc. White Paper, (July 10, 2000) (http://www.cyveillance.com/resources/7921S_Sizing_the_Internet.pdf)
16. Newman, M. E. J., Moore, C., Watts, D.: Mean-field solution of the small-world network model. Working Paper 99-09-066, Santa Fe Institute, Santa Fe, NM (Sept. 1999)
17. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the Web. Stanford Digital Library Project, Working Paper SIDL-WP-1999-0120, Stanford University, CA (1999)
18. Pirolli, P., Pitkow, P., Rao, R.: Silk from a sow's ear: Extracting usable structures from the Web. Proceedings of the ACM Conf. on Human factors in Computing, Vancouver, Canada (Apr. 13-18, 1996) 118-125
19. Rafiei, D., Mendelzon, A.: What is this page known for? computing Web page reputations. Proceedings of the 9th Int. WWW Conf., Amsterdam, The Netherlands (May 15-19, 2000)
20. Watts, D.: Small Worlds: The Dynamics of Networks between Order and Randomness. Princeton University Press, Princeton, NJ (1999)
21. Zhang, D., Dong, Y.: An efficient algorithm to rank Web resources. Proceedings of the 9th Int. WWW Conf., Amsterdam, The Netherlands (May 15-19, 2000)

Prefetching Tiled Internet Data Using a Neighbor Selection Markov Chain*

Yoo-Sung Kim[1], Ki-Chang Kim[2], and Soo Duk Kim[2]

[1] School of Information & Communication Engineering
[2] Department of Computer Science & Engineering
Inha University
Incheon 402-751, South Korea

Abstract. A large data file in the internet such as a map is served in small pieces, called tiles. To improve the service speed for such data, we can prefetch future tiles while the current one is being displayed. Traditional prefetching techniques examine the transition probabilities among the tiles to predict the next tile to be requested. However, when the tile space is very huge, and a large portion of it is accessed with even distribution, it is very costly to monitor all those tiles. In this paper, we propose a technique that captures the regularity in the tile request pattern by using an NSMC (Neighbor Selection Markov Chain) and predicts future tile requests based on it. The required regularity to use our technique is that the next tile to be requested is dependent on previous k movements (or requests) in the tile space. Map shows such regularity in a sense. Electronic books show a strong such regularity. We show how to build an NSMC and measure its prediction capability through experimentations.

1. Introduction

As the number of internet users grows, many kinds of data are being served on the internet. Some of them are static; that is, they are pre-generated before they are requested and held until a request comes. Some others are dynamic meaning they are generated on the fly when the request comes. Static data are easier for caching or prefetching because they are already available before the requests. They can be copied over from a web server and saved in a web cache server for future requests. Dynamic data, however, are not amenable for caching or prefetching. The actual data are generated anew each time a request is made. Since each request results in a new data, there is no point of caching or prefetching. Currently most cache servers do not cache or prefetch dynamic data. The cache server regards a request that contains an URL with arguments (starting with '?') as one for dynamic data and marks it as uncachable[1][1,12].

Still some others are static in the way the data are generated, but treated as if they are dynamic. An example is map data. It is pre-generated as a series of image files,

* This work is supported by Inha University in 2000.

[1] Given the same argument, however, the CGI program may generate identical results. In this case, it is possible to cache the data as indexed by the CGI program name and the arguments.

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 103–115, 2001.

called tiles, and stored in a map server waiting for requests; thus, it can be viewed as static data. However since the number of image files is very huge in general, it cannot be served at once in one piece. The client should specify which tile (or image file) he wants to view. If he wants to view several tiles, he has to make multiple requests. The specification for a tile is forwarded to the map server through CGI (Common Gateway Interface) protocol as if requesting for dynamic data. For the client, it seems that the CGI program has generated a new map portion, but the CGI program has merely picked up the corresponding pre-generated tile and returned. We define such data as map-like data, that is, a kind of data that has been pre-generated but is served by tiles via dynamic data service mechanism because of its huge size.

Currently most web cache servers do not cache or prefetch map-like data because it is served through CGI mechanism and thus regarded as dynamic data. However, the use of map-like data is increasing, as in maps, large PDF files, electronic book, etc., and fast transmission of them is becoming important. A tile in map-like data is typically larger than a regular html file, and users usually request a number of tiles that are nearby in their locations, sequentially in one session. Retrieving these many tiles, each of them being relatively large, over the internet will cause significant delays in transmission time.

In this paper, we propose a technique to speed up the transmission time of such map-like data. For static data like simple html files, the existing technique to deal with the transmission delay is caching or prefetching. Caching caches pages that have been requested previously for future use, in the client side or in a cache server located between the client and the web server. If a cached page is requested again, the client's web browser or the cache server can return it right away without going to the original web server. Prefetching tries to cache pages that have never been requested before but are expected to be requested soon [3,13]. We have chosen a prefetching technique to improve the service speed of map-like data. Caching popular tiles may also be helpful, but we assume there is a very large number of tiles for each map-like data, and that the requests are evenly distributed over the entire tile space except for a few popular tiles. For the popular tiles, caching can be done, but for the rest of tiles prefetching should be more effective.

Prefetching techniques for static data have been studied by numerous researchers (see Section 2). The main problem is how to predict the client's next request. If the current page is a first-timer, a page that has never been requested before, the only clue we have is the page itself. A cache server can parse the page to find all embedded links, and regard them as possible candidates for the next move. However, since the number of embedded links within a page is usually large, and since parsing a page is very time-consuming, this is not a practical solution. In fact, if the page is a first-timer, there is no effective way of predicting the next move without the help of the web server where the page resides. The web server knows the structure of the requested page, and by examining the reference pattern of it can tell the cache server what will be the next move. If the page has been requested several times before, the cache server may have collected a reference pattern by itself. Among the possible candidates, then, the cache server can predict which link will be followed next. If the predicted page is not cached yet, the cache server can now prefetch it from the web server. However, it is more likely that the predicted page is already cached in since the cache server must have noticed the importance of it.

We want to predict the next move for a map-like data. We want a cache server to predict the next move for such data without the help from a web server or a map

server even when the current page (or a tile in case of map-like data) is a first-timer. Previous techniques for prefetching cannot be applied directly to our problem, because they can deal with pages only if they have been requested repeatedly, or if they are first-timer pages for which web servers can give a help.

Our solution is based on two observations. We have observed that a tile has only a limited number of embedded links, and that among them only $2n$ links (for n-dimensional tile space) are important most of time. They correspond to the neighbor tiles of the current one. It is natural to expect that the next tile will be one of the neighbors of the current tile. For 1-dimensional tile space, there are two neighbors for each tile: left and right. For 2-dimensional space, there are four: left, right, up, and down. For n-dimensional space, there are $2n$ neighbors: two neighbors at each dimension. From this observation, we can form a candidate set for next move for any tile without parsing.

Our second observation is that if a set of tiles belonging to the same map-like data show similar transition patterns among them, we don't have to follow individual transition pattern for each tile. Instead, we can draw a single Markov chain, called Neighbor Selection Markov Chain (NSMC), for each map-like data that shows the general transition pattern of any tile belonging to it. Here, a state of this Markov chain does not represent a particular tile; instead, it represents k sequential movements that can be made, where k is determined by the characteristics of the particular map-like data. For example, a map-like data with 1-dimensional tile space may have a transition pattern such that the next tile is always the one in the right. For this data, it is appropriate to set k=1 because there is only one type of movement. In general, if the next movement depends on the previous k movements, the corresponding NSMC will have maximum $(2n)^k$ states. The cache server prepares these many states and builds edges based on the transition history[2]. With this Markov chain, the cache server can predict what will be the next movement from the current tile.

The remainder of the paper is organized as follows. Section 2 surveys related works on prefetching and dynamic data caching. Section 3 discusses the proposed NSMC model. Section 4 summarizes the experiment and its results. Finally, Section 5 gives the conclusions.

2. Related Researches

Prefetching between a web server and a client was studied by pioneers in [2,3,13]. In [13], a web server monitors the pages it owns to calculate transition probabilities for the embedded links. When a client requests a page, the server knows which link will be requested next with what probability. The server pushes a list of possible candidate links with their transition probabilities to the client. The client, then, prefetches selected candidates from the web server while the current page is being displayed. This technique requires both of the server and the client to be modified such that they behave correctly. Since we want to design a cache server that prefetches transparently without the cooperation from the web server or the client, this technique cannot be

[2] However, many of the states are never used, as shown in Section 4; thus the actual number of states is usually much smaller than $(2n)^k$.

applied directly to our case. A Markov algorithm is used in [2] to determine related objects. The server monitors the reference pattern of each client separately and pushes related objects to the client appropriately. We also use a Markov algorithm but to extract a general reference pattern, not individual pattern.

Prefetching can be performed also between a cache server and a web server. The authors in [6,11] suggest a technique in which the web server collects usage statistics (the transition probabilities of embedded links) of the pages and relays the information to the cache server. As in [2,13], the web server has to be modified. A technique in [5] excludes the need to modify web servers by forcing the cache server to prefetch all embedded links within the currently requested page. To do this, however, the cache server has to parse each page to detect embedded links, and the overhead of fetching all of them is huge. Prefetching between a client and a cache server was discussed in [10]. The cache server collects the usage statistics for all pages it caches, and in order to do this, it parses each page remembering embedded links, and as requests are made, collects information about which link is referenced when and how often. The cache server pushes this information to the client, and the client decides which links will be prefetched. The client has to be modified to perform this prefetching.

Using Markov chains to predict the client's next request is studied in [9,14]. In [9], the expected number of accesses to each document within a specified time period from the current document is computed using a Markov chain. The highly scored documents will become possible candidates for prefetching. In [14], a Markov chain is built to store the transition probabilities among pages. Combined with the client's reference pattern, it can provide several value-added services like tour generation or personalized hubs/authorities generation. Both studies, however, try to compute exhaustive transition probabilities for all documents or pages, which is too expensive to apply to tile data.

The authors in [15] suggest a technique to speed up the retrieval time of geographical data where the query result is usually very huge in size. Traditionally, to cope with the huge size, the database application program executes an iterator-based algorithm as follows.

```
Result result = database.query(condition);
    while (result.more()){
       doSomething(result.get());
     }
```

The more() and get() are remote procedures to the database server, and calling them repeatedly incurs a high overhead. To reduce the overhead, they suggest to use a cache server inside the client machine that will intercept the database query, open a persistent connection to the database server by itself, and supplies the query result to the client as if it is the database server. The client does not know about the intervening cache server and, through the same iterator-based interface, requests and receives the result. Since the cache server prefetches query results while the client is processing the current portion of the result, the retrieval time of whole results is improved. However, again, the client has to be modified, and the cache server in this case knows exactly what the client will request next: it does not have to predict.

3. Prefetching Tiles over the Internet

3.1 Basic Idea

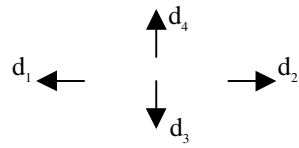
The references for tiles of a map-like data seem to show some spatial locality. When a tile has been requested, there is a high chance that one of the neighboring tiles will be requested next. The problem is which one, among the neighbors, will be requested next. The traditional prefetching techniques predict based on the reference history. They try to remember which neighbor was selected how many times from which tile. However, it is not easy to collect and maintain this information for all tiles. Furthermore, since we assume the number of tiles is very huge, there should be many tiles that are never referenced before. For these tiles, when they are referenced for the first time, how can we predict the next tile? In regular web data, the proportion of such first-timer data is about 30%[4]. We believe the proportion will be similar in tile data, and without handling properly these first-timer tiles, the prefetching would not achieve desired result.

Our observation is that if tiles belonging to the same map-like data show similar neighbor selection probabilities, we don't have to follow the reference history of each tile separately. Instead, we can try to capture the representative neighbor selection probabilities that can be applied to a general tile. Once we capture them, we can predict the next neighbor for any tile whether it has been referenced before or not.

3.2 Building a Neighbor Selection Markov Chain

In order to obtain representative neighbor selection probabilities, we build a Neighbor Selection Markov chain, NSMC. An NSMC has a number of states and connecting edges. A state of NSMC is a sequence of direction vectors that shows a tile selection history, while an edge is the probability to reach the target neighboring tile from the current one. For n-dimensional tile space, there are $2n$ neighbors for each tile because it has 2 neighbors, backward and forward, at each dimension. Diagonal neighbors can be expressed by the combination of these basic neighbors. Let's denote direction vectors for n-dimensional tile space as $d_1, d_2, ..., d_{2n}$. Direction vector d_i and d_{i+1} respectively represents the backward and forward movement to reach the two neighboring tiles at dimension $\frac{i+1}{2}$, for odd i. Also let's denote a tile at coordinates $(i_1, i_2, ..., i_n)$ as $t_{t1,i2,...in}$.

A tile selection history can be expressed as a sequence of direction vectors. For example, assuming 2-dimensional tile space, a tile selection history, t_{33}, t_{34}, t_{44}, can be expressed as (d_4, d_2). The direction vectors at dimension 2 are shown below.



Now given an NSMC, predicting a next tile means deciding a next direction vector from the current state in an NSMC. The current state is defined as a sequence of direction vectors selected so far. To limit the size of NSMC, we restrict that the state can contain maximum k previous direction vectors, where k being a characteristic number specific to each map-like data. For n-dimensional tile space and k direction vectors for each state, the number of possible states is $(2n)^k$. To build an NSMC, therefore, we need to build a Makov chain for these many states in maximum. The algorithm to build an NSMC is in Section 4.2.

3.3 Examples

Suppose a map-like data that can be modeled by a 1-dimensional tile space with k=1. Electronic books will be an example of such data. The tiles (pages) of an electronic book are connected linearly; therefore n=1. If we model them with k=1, that means we predict the next tile based only on the last move we made to reach the current tile. That is we compute the next direction vector based on the previous single direction vector. Since n=1, there are only 2 possible direction vectors: d_1 for backward movement, and d_2 for forward movement. For electronic books, there is a high chance that the next direction vector will inherit the previous direction vector, because we can guess if the client was reading the books in forward or backward direction, he would continue reading in that direction for a while. Since n=1 and k=1, the total number of states is $(2n)^k$=2. The edge probabilities will be adjusted as references for this data are processed. The NSMC for this case is in Fig.1. The transition probabilities are not shown, but the loop-back probabilities (d_2->d_2 and d_1->d_1 edges) will be much higher than the crossing-over probabilities (d_2->d_1 and d_1->d_2 edges).

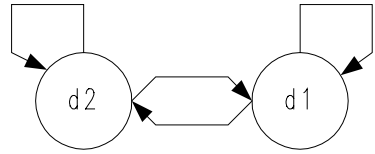


Fig. 1. An NSMC with n=1 and k=1.

To show an NSMC with k > 1, let's assume the client scans tiles along a Hilbert curve. An NSMC with n=2 and k=3 can be used to predict the client's movement. That is, without knowing about Hilbert curves, an NSMC can capture the regularity in client's movement. A Hilbert curve is being used to map a n-dimensional space on to a 1-dimensional space such that the adjacent objects in a n-dimensional space remain as adjacent as possible in a 1-dimensional space[8]. A Hilbert curve at level 3 is in Fig. 2. At level x, the number of tiles is 4^x, and the Hilbert curve at level x can be

drawn by connecting four Hilbert curves at level (x -1) with appropriate rotation and change of sense –- the two upper quadrants with no rotation and no change of sense, while the lower left quadrant being rotated clockwise 90 degree, and the upper right quadrant being rotated anti-clockwise 90 degree[8]. As the level goes up, the curve gets complex; however, the 4 cups in Fig. 3 are the basic building blocks. A Hilbert curve at any level can be represented as a combination of these 4 cups. Each cup represents a sequence of three consecutive movements that happens repeatedly.

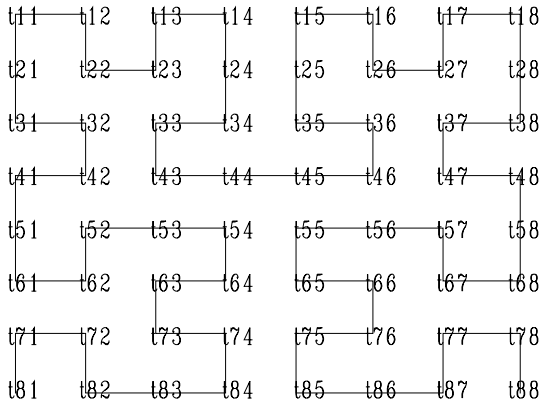


Fig. 2. A Hilbert curve at level 3.

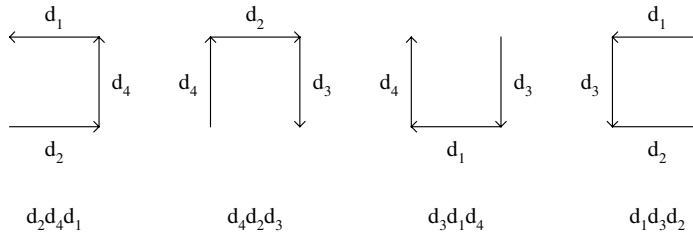


Fig. 3. Basic 4 cups of Hilber curves.

The Hilbert curve in Fig. 2 can be captured by an NSMC in Fig 5. To obtain the NSMC, we first draw a state diagram as in Fig.4 that shows how the 4 basic cups are connected to each other to form the movements in Fig. 2. Fig. 5 shows that we can predict the next move of a client when we know the previous three consecutive movements. For example, at a state denoted by ($d_3d_2d_4$), the next state is always ($d_2d_4d_2$). Since the last movement seen at the state ($d_2d_4d_2$) is d_2 (the rightmost one), we conclude that, at state ($d_3d_2d_4$), the next move is always d_2. This is the case when t_{12}, t_{23}, t_{23}, and t_{13} are selected in a row, in Fig. 2. The next movement

predicted is d_2; therefore, the next tile should be t_{14}, which is correct in the Hilbert order. The same argument can be made for the sequence of t_{16}, t_{26}, t_{27}, and t_{17}.

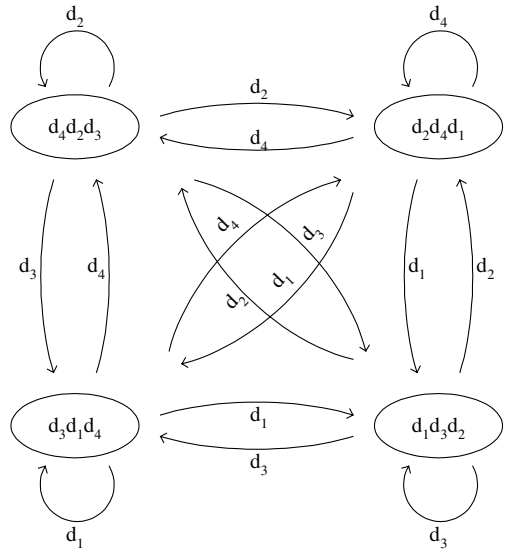


Fig. 4. A state diagram for the Hilber curve in Fig. 2.

4. Experiments

In section 3, we have explained how an NSMC can be built given a sequence of tile requests. Now we examine the effectiveness of an NSMC in predicting the client's next request. To force regularity in the client's tile requests, we assume the client request tiles in an order dictated by a Hilbert curve as explained in Section 3. In reality, there should be some randomness and regularity intermixed in a client's tile request sequence. If it is strictly random, prediction is not possible. Otherwise, the NSMC will capture the regular portion of the pattern. The negative influence of the random portion of the pattern on the predictability of an NSMC can be measured by incorporating randomness in a given request sequence. The experiment was carried out in three steps: generating a Hilbert ordering, generating NSMCs, and measuring the prediction performance of them.

4.1 Generating a Hilbert Ordering

We have generated Hilbert numbers for all the grid points in a square grid of size 128*128, and saved them in a table. This table is used to generate a sequence of tile requests in Hilbert order. The algorithm is given in Algorithm 1. The algorithm to compute a Hilbert number given x and y coordinates was taken from [8].

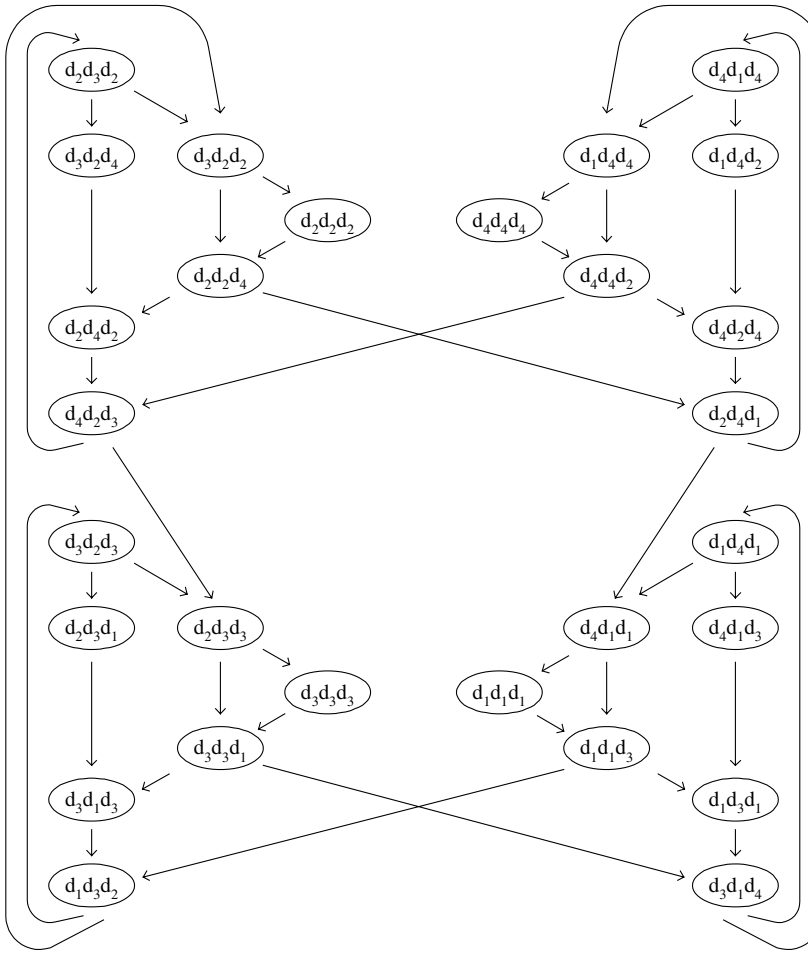


Fig. 5. NSMC for the Hilbert curve in Fig. 2.

```
Algorithm 1 : Computing a Hilbert ordering
Input: x, y, side
Output: Hilbert numbers corresponding to the grid
points in a square grid with size side*sideMethod:
for(x=0;x<side;x++)
   for(y=0;y<side;y++){
       hnum = compute Hilbert number from (x,y);
       hilbert_ordering[hnum].x = x;
       hilbert_ordering[hnum].y = y;
   }
```

Three NSMCs were generated for k =1, 2, and 3. As in Section 3, k represents the number of previous movements kept in an NSMC state. With bigger k, we can

predict more accurately, however the number of possible NSMC states also grows exponentially[3]. The algorithm is given in Algorithm 2. In the algorithm, S_j is the j^{th} state, and M_i is the direction vector to move from hilbert_ordering[i-1] to hilbert_ordering[i]. The path of S_j, the recent k movements to reach S_j, is represented by $m_{j0}m_{j1}...m_{j(k-1)}$.

Basically, we generate a sequence of tile requests in the Hilbert order computed in Algorithm 1, and build an NSMC based on them. First $k+1$ requests are converted to k movements and stored in state 0. From then on, each movement will cause either the formation of a new state, if its path has not appeared before, or the search for a previous state that has this path. In both cases, a link is added from the current state to the next state, if there is not one yet. For a new link, the count is initialized; otherwise, the counter is increased by 1. The ratio of the link counts over the visiting counts of the source state is the transition probability for it. Fig. 6 shows the resulting NSMCs. For the case of k =3, we have a graphic form in Fig. 5, and omitted it.

```
Algorithm 2 : Building an NSMC
Input: hilbert_ordering[], k
Output: NSMC for k
Method:
  m00m01...m0(k-1) =
    direction vectors to move from lbert_ordering[0] to
    ilbert_ordering[k];
  curr_state=0;
  next_state = 1;
  for(i=k+1; i< side*side; i++){
     compute next move, Mi ;
     new_path = mcurr_state,0mcurr_state,1...mcurr_state,k-1Mi ;
     if (there is a previous state, Sj , whose path is
     same as new_path){
        add Sj to the child list of Scurr_state ;
        curr_state =j;
     } else{
        save new_path as the path for next_state;
        add next_state to the child list of curr_state;
        curr_state = next_state;
        next_state++;
     }
  }
```

[3] The maximum possible number of states grows by $(2n)^k$. However, the actual NSMC does not use all the states. For Hilbert curve, they were 4, 12, and 28, for k=1, 2, and 3, respectively.

state	path (k=1)	child list
0	d4	1 3 0
1	d2	2 1 0
2	d3	1 3 2
3	d1	0 3 2

state	path (k=2)	child list
0	d4 d2	1 4
1	d2 d3	2 11 9
2	d3 d2	3 4 1
3	d2 d2	4 3
4	d2 d4	5 0
5	d4 d1	6 7 8
6	d1 d4	0 10 5
7	d1 d1	8 7
8	d1 d3	9 2
9	d3 d1	6 8
10	d4 d4	0 10
11	d3 d3	9 11

Fig. 6. NSMCs for k=1 and k=2.

4.3 Measuring the Prediction Performance

To measure the NSMCs' prediction performance, we have generated the same sequence of tile requests as in Section 4.2 and feed them to the three NSMCs. This time, however, we use the NSMC to predict the next request for each request and compare it to the actual one. The number of correct guessing is accumulated. We also allow the NSMC to select more than one candidate to see how the performance improves. The algorithm is given in Algorithm 3. In the algorithm, *p* is number of candidates the NSMC can select. We have measured the performance of three NSMCs in Section 4.2 for *p*=1, 2, and 3. The result is in Fig. 7.

The simplest case is when k=1 and p=1. The NSMC in this case is very small (with only 4 states) and cheap to use because it prefetches only one candidate; however the hit rate is already around 40%. The NSMC is correctly guessing the next move for 6596 requests out of total 16384 requests. With k=3 and p=2, we can predict the client's behavior perfectly except in a few initial requests. The size of NSMC in this case is 28, still manageable. Even if we limit the number of candidates to 1, the hit rate is still quite high, 71%. By increasing k, we can further improve the accuracy of prediction.

```
Algorithm 3 : Computing prediction performace of NSMC.
Input: hilbert_ordering[], k
Output: the prediction performance of NSMC for k
Method:
curr_state = 0;
hit = 0;
for(i=k+1; i< side*side; i++){
```

```
        compute M_i;
        select p children from curr_state;
        compute direction vector m_1,m_2,...m_p to reach hose
        children;
if M_i is in (m_1,m_2,...m_p) hit++;
}
```

	k = 1	k = 2	k=3
p = 1	6596/16384, 40%	11468/16384, 70%	16382/16384, 99%
p = 2	8962/16384, 55%	14659/16384, 89%	16382/16384, 99%
p = 3	11704/16384, 71%	16382/16384, 99%	16382/16384, 99%

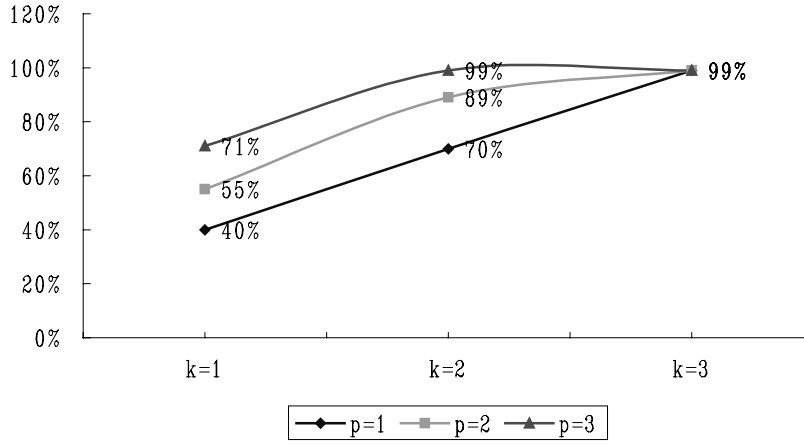


Fig. 7. Prediction Performance of NSMCs

5. Conclusions

When there exists regularity in the access pattern for internet data object, capturing and exploiting it for the purpose of prefetching is much cheaper than collecting and maintaining the reference histories of individual objects. Tiles for map-like data, e.g. maps, large PDF files, electronic books, etc., show such regularity in various degrees. By capturing the regularity, we can predict what tile will be requested next by the client, and by prefetching those tiles ahead of time, the transmission delay seen by the client will be reduced considerably.

In this paper, we have shown that an NSMC(Neighbor Selection Markov Chain) can be used to capture such regularity that may exist in the tile request pattern. The

required regularity is that the next tile can be predicted given the previous k tile requests. We have explained how to build an NSMC and measured its performance for a request pattern that follows a Hilbert curve. The experiments show, with $k=3$, the NSMC predicts future request with 71% accuracy for $p=1$ and 99% accuracy for $p=2$, where p is the number of candidates the NSMC can select.

References

1. G. Banga, F. Douglis, and M.Rabinovish, "Optimistic deltas for WWW latency reduction," Proc. of USENIX'97, pp289-303, Jan. 1997
2. Azer Bestavros and Carlos Cunha, "Server-initiated document dissemination for the www," IEEE Data Engineering Bulletin, September 1996.
3. Pei Cao, Edwad W. Felten, Anna R. Karlin, and Kai Li, "A study of integrated prefetching and caching strategies," Proc. 1995 ACM SIGMETRICS, pp 188-197, May 1995.
4. B. Duska, D. Marwood, and M. Feeley, "The measured access characteristics of world wide web client proxy caches," Proc. of USENIX Symposium of Internet Technologies and Systems(USITS), pp. 23-35, Dec. 1997.
5. Wcol Group, "WWW collector – the prefetching proxy server for www," http://shika.aist-nara.ac.jp/products/wcol/wcol.html, 1997.
6. James Gwertzman and Margo Seltzer, "The case for geographical push-caching," Proceedings of the Fifth Workshop on Hot Topics in Operating Systems, 1995.
7. V. Holmedahl, B. Smith, and T.Yang, "Cooperative caching of dynamic content on a distributed web server," Proc. of Seventh IEEE International Symposium on High Performance Distributed Computing(HPDC-7), pp243-250, 1998
8. H. V. Jagadish, "Linear Clustering of Objects with Multiple Attributes," Proc. of ACM SIGMODD '90 Intl. Conf. on Management of Data, May 1990.
9. A. Kraiss and G. Weikum, "Integrated document caching and prefetching in storage hierarchies based on Markov-chain predictions," The VLDB Journal 7(7)(1998) 141-162.
10. Tong sau Loon and Vaduvur Bharghavan, "Alleviating the latency and bandwidth problems in www browing", Proc. of the 1997 USENIX Symposium on Internet Technology and Systems, Dec. 1997 .
11. Evangelos P. Markatos and Catherine E. Chronaki, "A top-10 approach to prefetching on the web," Technical report, Technical Report No. 173, ICS-FORTH, Heraklion, Crete, Greece, August 1996. URL http://www.ics.forth.gr/proj/arch-vlsi/www.html
12. J. Mogul, F. Douglish, A. Feldmann, and b. Krishnamurthy, "Potential benefits of delta-encoding and data compression for HTTP," Proc. of SIGCOMM'97, pp.181-94, Sept. 1997.
13. Venkata N. Padmanabhan and Jeffrey C. Mogul, "Using predictive prefetching to improve world wide web latency," ACM SIGCOMM Computer Communication Review, July 1996.
14. Ramesh R. Sarukkai, "Link prediction and path analysis using Markov chains," Proc. of 9th International world wide web conference, May 15-19, 2000.
15. Edward P.F. Chan and Koji Ueda, "Efficient query result retrieval over the web," Proc. of the 7th International Conference on Parallel and Distributed Systems, July 2000.

A General Adaptive Cache Coherency-Replacement Scheme for Distributed Systems

Jose Aguilar[1] and Ernst Leiss[2]

[1] CEMISID, Dpto. de Computacion, Facultad de Ingenieria, Universidad de Los Andes, Merida 5101, Venezuela
aguilar@ing.ula.ve
[2] Department of Computer Science, University of Houston, Houston, TX 77204-3475, USA
coscel@cs.uh.edu

Abstract. We propose an adaptive cache coherence-replacement scheme for distributed systems that is based on several criteria about the system and applications, with the objective of optimizing the distributed cache system performance. We examine different distributed platforms (shared memory systems, distributed memory systems, and web proxy cache systems) and the potential of incorporating coherency-replacement issues in the cache memory management system. Our coherence-replacement scheme assigns a replacement priority value to each cache block according to a set of criteria to decide which block to remove. The goal is to provide an effective utilization of the distributed cache memory and a good application performance

1 Introduction

The performance of distributed caching mechanisms is an active area of research [2, 3, 4, 5, 7, 8, 10]. A distributed cache memory is the simplest cost-effective way to achieve a high-speed memory hierarchy. A cache provides, with high probability, instructions and data needed by the local CPU at a rate that is more in line with the local CPU's demand rate. Many studies have examined policies for cache replacement and cache coherence; however, these studies have rarely taken into account the combined effects of policies [2, 5]. In this paper we propose an adaptive cache coherence-replacement scheme for distributed systems. Our approach combines classical coherence protocols (write-update and write-invalid protocols) and replacement policies (LRU, LFU, etc.) to optimize the overall performance (based on criteria such as network traffic, application execution time, data consistence, etc.). This work is based on previous work we have done on cache replacement mechanisms which have shown that adaptive cache replacement policies improve the performance of computing systems [1]. The cache coherence mechanism is responsible for determining whether a copy in the distributed cache system is stale or valid. At the same time, it must update the invalid copies when a given site requires a block. We study the impact of our scheme in different distributed systems: shared-memory systems, distributed-memory systems, and web proxy systems.

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 116–125, 2001.

2. Theoretical Aspects

2.1 Coherence Problem

Distributed cache systems provide decreased latency at a cost: every cache will sometimes provide users with *stale* pages. Every local cache must somehow update pages in its cache so that it can give users pages which are as fresh as possible. Indeed, the problem of keeping cached pages up to date is not new to cache systems: after all, the cache is really just an enormous distributed file system, and distributed file systems have been with us for years. In conventional distributed systems terminology, the problem of updating cached pages is called *coherence* [2, 4, 5, 7, 12].

Specifically, the cache coherence problem consists of keeping a data element found in several caches current with each other and with the value in main memory (or local memories). A *cache coherence protocol* ensures the data consistency of the system: the value returned by a read must always be the last value written to that location. There are two classes of cache coherence protocols [12]: write-invalidate and write-update. In a *write-invalidate* protocol, a write request to a block invalidates all other shared copies of that block. If a processor issues a read request to a block that has been invalidated, there will be a coherence miss. That is, in *write-invalidate* protocols whenever a processor modifies its cache block, a *bus invalidation signal* is sent to all other caches in order to invalidate their content. In a *write-update* protocol on the other hand, each write request to shared data updates all other copies of the block, and the block remains shared. That is, in *write-update* protocols a copy of the new data is sent to all caches that share the old data. Although there are fewer read misses for a write-update protocol, the write traffic on the bus is often so much higher that the overall performance is decreased. A variety of mechanisms have been proposed for solving the cache coherence problem. The optimal solution for a multiprocessor system depends on several factors, such as the size of the system (i.e., the number of processors), etc. The main types of coherence mechanisms are [12]: Snooping Coherence, Directory Coherence, and Software Cache Coherence Mechanism.

2.2. Replacement Policy Problem

A replacement policy specifies which block should be removed when a new block must be entered into an already full cache; it should be chosen so as to ensure that blocks likely to be referenced in the near future are retained in the cache. The choice of replacement policy is one of the most critical cache design issues and has a significant impact on the overall system performance. Common replacement algorithms used with such caches are [1, 3, 6, 8, 9]: First In-First Out (FIFO), Most Recent Used (MRU), Least Recently Used (LRU), Least Frequently Used (LFU), Least Frequently Used (LFU)-Aging, Greedy Dual Size (GDS), Frequency Based Replacement (FBR), Random (RAND), Priority Cache (PC), Prediction.

In general, the policies anticipate future memory references by looking at the past behavior of the programs (program's memory access patterns). Their job is to identify

a line/block (containing memory references) which should be thrown away in order to make room for the newly referenced line that experienced a miss in the cache.

3. An Adaptive Coherence-Replacement Policy

Normally, user cache access patterns affect cache replacement decisions while block characteristics affect cache coherency decisions. Therefore, it is reasonable to consider replacing cache blocks that have expired or are closed to expiring because their next access will result in an invalidation message. In this way, we propose a cache coherence-replacement mechanism that incorporates the state information into an adaptive replacement policy. The basic idea behind the proposed mechanism is to combine a coherence mechanism with our adaptive cache replacement algorithm [1]. Our adaptive cache coherence-replacement mechanism exploits semantic information about the expected or observed access behavior of particular data shared objects on the size of the cache items, and the replacement phase employs several different mechanisms, each one appropriate for a different situation. Since our coherence-replacement is provided in software, we expect the overhead of providing our mechanism to be offset by the increase in performance that such a mechanism will provide. We incorporate the additional information about a program's characteristics, which is available in the form of the cache block states, in our replacement system. Our system can be applied to different distributed systems independent of the coherence protocol. We assume that if a place has an invalid copy of a block, a request to this block is a miss access. In this way, we guarantee to search for a valid copy of the block. We assume three types of target systems:

3.1 Cache Coherence-Replacement in a Shared Memory Multiprocessor

In a shared-memory multiprocessor system, *local caches* are used to reduce memory access latency and network traffic [8, 12]. Each processor is connected to a fast memory 'backed up' by a large (and slower) main memory. This configuration enables processors to work on local copies of main memory blocks, greatly reducing the number of memory accesses that the processor must perform during program execution. Although local caches improve system performance, they introduce the *cache coherence problem*: multiple cached copies of the same block of memory must be consistent at any time during a run of the system. In general, each cache block can be in one of the following four states:

Invalid: a stale copy.
Shared: multiple copies of the block exist.
Exclusive: only one processor has a copy of the block.
Modified: the processor has the only valid copy of the block in cache.

We can use both coherence protocols (write-invalid and write-update). In this case, our adaptive cache coherence-replacement mechanism is as follows:

1. If *write miss* (if the processor has a copy of the block, it is invalid) then
   1.1 Search for a valid copy (shared memory or remote cache memory). A *read-miss* request is sent to the system
   1.2 If cache is full, choose a replacement policy according to a *decision system*.
   1.3 Receive a valid copy
   1.4 Modify block (critical section)
   1.5 Call coherence protocol (write-invalidate or write-update protocol)
   1.6 Change state to modified (if we use write-invalidate protocol) or shared (if we use write-update protocol) or exclusive (if it has the only copy in the system)
   1.7 Change state of all other copies of this block to invalid (if we use write-invalidate protocol) or shared (if we use write-update protocol)
2. If *read miss* then
   2.1 Search for a valid copy (shared memory or remote cache memory). A *read-miss* request is sent to the system
   2.2 If cache is full, choose a replacement policy according to a *decision system*.
   2.3 Receive a valid copy
   2.4 Change state of the different copies of the block to shared or exclusive (if it has the only copy on the system)
   2.5 Read block
3. If *write hit* then
   3.1 Modify block (critical section)
   3.2 Call coherence protocol (write-invalidate or write-update protocol)
   3.3 Change state to modified (if we use write-invalidate protocol) or shared (if we use write-update protocol) or exclusive (if it has the only copy on the system)
   3.4 Change state of all other copies of this block to invalid (if we use write-invalidate protocol) or shared (if we use write-update protocol)
4. If *read hit* then
   4.1 Read block

3.2 Cache Coherence-Replacement in a Distributed Memory Multiprocessor

In this case, we study a distributed memory multiprocessor (DM) in which each processor is associated with a private cache. Local cache memory is the simplest cost-effective way to achieve a high-speed memory hierarchy in a distributed memory system. A local cache provides, with high probability, instructions and data needed by the local CPU at a rate that is more in line with the CPU's demand rate [2, 12]. In this case, each cache block can be in one of the following state:

Invalid: a stale copy.

Shared_up: multiple copies of the block exist and all memory copies are up-to-date.
Shared_nup: multiple copies of the block exist and not all memory copies are up-to-date.
Exclusive_up: only one processor has a copy of the block, and the local memory copy is up-to-date.
Exclusive_nup: only one processor has a copy of the block, and the local memory copy is not up-to-date.
Modified: the processor has the only valid copy of the block and all memory copies are stale.

In our approach, we assume a protocol which guarantees that the local memory has the same version of a given block as its cache memory. That means, the differentiation between Shared_up and Shared_nup is not necessary (only a Shared state is needed); similarly for Exclusive_up and Exclusive_nup, where only an Exclusive state is required. In addition, when we update the state of a block at a given site (according to the coherence protocol), both block copies are updated (local memory and its cache memory) if the cache memory has a copy. With these assumptions, the adaptive cache coherence-replacement mechanism for this case is the same as the previous one.

3.3 Cache Coherence-Replacement in a Web Proxy Cache

The growth of the Internet and the WWW has significantly increased the amount of online information and services available. However, the client/server architecture employed by the current Web-based services is inherently unscalable. Web caches have been proposed as a solution to the scalability problem [3, 4, 5, 7, 10, 13]. Web caches store copies of previously retrieved objects to avoid transferring those objects in response to subsequent requests. Web caches are located throughout the Internet, from the user's browser cache through local proxy caches and backbone caches, to the so-called reverse proxy caches located near the origin of the content. Client browsers may be configured to connect to a proxy server, which then forwards the request on behalf of the client. All Web caches must try to keep cached pages up to date with the master copies of those pages, to avoid returning stale pages to users. There are strong benefits for the proxy to cache popular requests locally. Users will receive cached documents more quickly. Additionally, the organization reduces the amount of traffic imposed on its wide-area Internet connection.

Because a cache server has a fixed amount of storage, the server needs a cache replacement mechanism [3, 5]. Recent studies on web workload have shown tremendous breadth and turnover in the popular object set-the set of objects that are currently being accessed by users [13]. The popular object set can change when new objects are published, such as news stories or sports scores, which replace previously popular objects. We should define cache replacement policies based on this workload characterization. In addition, a cache must determine if it can service a request, and if so, if each object it provides is fresh. This is a typical question to be solve with a cache coherence mechanism. If the object is fresh, the cache provides it directly, if not, the cache requests the object from its origin server.

Our adaptive coherence-replacement mechanism for Web caches is based on systems like Squid [11], which caches Internet data. It does this by accepting requests for objects that people want to download and by processing their requests at their sites. In other words, if users want to download a web page, they ask Squid to get the page for them. Then Squid connects to the remote server and requests the page. It then transparently streams the data through itself to the client machine, but at the same time keeps a copy. The next time someone wants that same page, Squid simply reads it from its disks, transferring the data to the client machine almost immediately (Internet caching). Normally, in Internet caching cache hierarchies are used. The Internet Cache Protocol (ICP) describes the cache hierarchies. The ICP's role is to provide a quick and efficient method of intercache communication, offering a mechanism for establishing complex cache hierarchies. ICP allows one cache to ask another if it has a valid copy of a object. Squid ICP is based on the following procedure [11]:

1. Squid sends an ICP query message to its neighbors (URL requested)
2. Each neighbor receives its ICP query and looks up the URL in its own cache. If a valid copy exists, the cache sends ICP_HIT, otherwise ICP_MISS
3. The querying cache collects the ICP replies from its peers. If the cache receives several ICP_HIT replies from its peers (neighbors), it chooses the peer whose reply was the first to arrive in order to receive the object. If all replies are ICP_MISS, Squid forwards the request to the neighbors of its neighbors, until to find a valid copy.

Neighbors refer to other caches in a hierarchy (a parent cache, a sibling cache or the origin server). Squid offers numerous modifications to this mechanism, for example: i) Send ICP queries to some neighbors and not to others, ii) Include the origin sever in the ICP "ping" so that if the origin servers reply arrives before any ICP-hits, the request is forward there directly, iii) Disallow or require the use of some peers for certain requests. In this case, each cache block can be in one of the following states:

Invalid: a stale copy.

Normally, there is only one state because the users typically do not write. Then, the adaptive cache coherence-replacement mechanism is as follows:

1. If *read miss* then
   1.1 Search for a valid copy (using the ICP). A read-miss request is sent using the ICP
   1.2 If cache is full, choose a replacement policy according to a *decision system*.
   1.3 Receive a valid copy
   1.4 Read block
2. If *read hit* then
   2.1 Read block

3.4 Our Generic Replacement Subsystem

Typically, a cache replacement technique must be evaluated with respect to an offered workload that describes the characteristics of the requests to the cache. Of particular interest are patterns in the objects referenced and the relationships among accesses. Workload is sufficiently complicated that we can use other types of information to try to solve this problem. Thus, we define a set of parameters that we can use to select the best replacement policy in a dynamic environment:

- Information about the system: Workload, Bandwidth, Latency, CPU Utilization, Type of system (Shared memory, etc.)
- Information about the application: Information about the data and cache block or objects (Frequency, Age, Size, Length of the past information (patterns), State (invalid, shared, etc.)), Type an degree of access pattern on the system (High or low spatial locality (SL), High or low temporal locality (TL)).
- Other information: Cache conflict resolution mechanism, Pre-fetching mechanism.

An optimal cache replacement policy would know the future workload. In the real world, we must develop heuristics to approximate ideal behavior. For each of the policies we listed in section 2.2, we define the information that is required by them:

- LFU: reference count.
- LRU: the program's memory access patterns.
- Priority Cache: information at runtime or compile time (data priority bit by cache/block).
- Prediction: a summary of the entire program's memory access pattern.
- FBR: the program's memory access patterns and organization of the cache memory.
- MRU: the program's memory access patterns.
- FIFO: the program's memory access patterns.
- GDS: size of the objects, information to calculate the cost function, reference count.
- Aging approaches: GDS-aging: GDS age factor or LFU-aging: LFU age factor.

We define one expression, called the *key value*, to define the priority of replacement of each block/object. According to this value, the system chooses the block with higher priority to replace (low key value). The key value is defined as:

$$\text{Key-Value} = (CF+A+FC)/S + \text{cache factor} \quad (1)$$

where,

- FC is the frequency/reference count, that is the number of times that a block has been referenced,
- A is the age factor,

- S is the size of the block/object,
- CF is the cost function that can include costs such as latency or network bandwidth.

The first part of Equation (1) is typical for the GDS, LRU and LFU policies (using information about objects to reference and not about cache blocks). The cache factor is defined according to the replacement policy used:

- LFU: blocks with a high frequency count have the highest cache factor.
- LRU: the least recently used block has the highest cache factor.
- Priority Cache: defined at runtime or compile-time.
- Prediction: the least used block in the future has the highest cache factor.
- FBR: the least recently used block has the highest cache factor.
- MRU: the most recently used block has the highest cache factor.
- FIFO: the block at the head of the queue has the highest cache factor.
- GDS: not applicable.
- Aging approaches: FC/A, with a reset factor that restarts this value after a given number of ages or when the age average is more than a given value.

The coherence-replacement policy defines the cache factor so that: blocks in invalid state have the highest priority to be chosen to replace. Otherwise, blocks in shared states must be chosen to replace, then blocks in exclusive states, and finally, blocks in modified states. If there are several blocks in a particular state, we use the replacement policy specified in our *decision system* [1]. The *decision system* is composed of a set of rules to decide the replacement policy to use. Each rule selects a replacement policy to apply according to different criteria:

If *TL is high and the system's memory access pattern is regular* then
    Use a LRU replacement policy
If *TL is low and the system's memory access pattern is regular* then
    Use a LFU replacement policy
If *TL is low and the system's memory access pattern is large* then
    Use a MFU replacement policy
If *we require a precise decision using a large system's memory access pattern history* then
    Use a Prediction replacement policy
If *objects/blocks have variable sizes* then
    Use a GDS replacement policy
If *a fast decision is required* then
    Use a RAND replacement policy
If *there is a large number of LRU candidate blocks* then
    Use a FBR replacement policy
If *SL is high* then
    Use a hybrid FBR + GDS replacement policy
If *the system's memory access pattern is irregular* then

Use an age replacement policy
If *the Local CPU utilization is low* then
Choose a process to suspend (In this case, we can use a PC policy to select the process) and call the algorithm again for this reduced set of processes.

For all other situations, choose randomly a replacement policy. The CPU utilization criterion avoids the starvation or thrashing problem on the system. In general, these rules are based on the average of the different criteria in the system. We can take into account the victim's information (the process that has been selected to expel its page) with the next rule:

If *the victim's SL or TL is high* then
Lock this page % Choose another page
And call the decision system again with this smaller problem.

If we don't find a page to expel according to the last rule, we choose the first victim process that we had selected and we suspend this process. In the case of web proxy systems, we don't use the last two rules because they are very specific for multiprocessing systems.

4. Conclusions

The goal of this research was to formulate an overarching framework subsuming various cache management strategies in the context of different distributed platforms. We have proposed an adaptive coherence-replacement policy. Our approach includes additional information/factors such as frequency of block use, state of the blocks, etc., in replacement decisions. It takes into consideration that coherency and replacement decisions affect each other. This adaptive policy system must be validated by experimental work in the future, for example, using the Squid open source proxy cache [11]. In general, we plan to do a prototype implementation of our techniques and study their impact on the quality of the cache management, taken into consideration the additional cost (in time and space) our approach requires.

Acknowledgment

Jose Aguilar was supported by a CONICIT-Venezuela grant (subprograma de pasantías postdoctorales).

References

1. Aguilar J., Leiss E. A Proposal for a Consistent Framework of Dynamic/Adaptive Policies for Cache Memory Management, Technical Report, Department of Computer Sciences, University of Houston, (2000).
2. Cho S., King J., Lee G. Coherence and Replacement Protocol of DICE-A Bus Based COMA Multiprocessor, Journal of Parallel and Distributed Computing, Vol. 57 (1999) 14-32.

3. Dilley J., Arlitt M. Improving Proxy Cache Performance: Analysis of Three Replacement Policies, IEEE Internet Computing, November, (1999) 44-50.
4. Krishnamurthy B., Wills C. Piggyback Server Invalidation for Proxy Cache Coherency, Proc. 7th Intl. World Wide Web Conf., (1998) 185-193.
5. Krishnamurthy B., Wills C. Proxy Cache Coherency and Replacement-Towards a More Complete Picture, IEEE Computer, Vol. 6, (1999) 332-339.
6. Lee D., Choi J., Noh S., Cho Y., Kim J., Kim C. On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies, Performance Evaluation Review, Vol. 27 (1999). 134-143.
7. Liu C., Cao P. Maintaining Strong Cache Consistency in the WWW, Proc. 17th IEEE Intl. Conf. on Distributed Computing Systems, (1997).
8. Mounes F., Lilja D. The Effect of Using State-based Priority Information in a Shared-Memory Multiprocessor Cache Replacement Policy, IEEE Computer, Vol. 2 (1998) 217-224.
9. Obaidat M., Khalid H. Estimating NN-Based Algorithm for Adaptive Cache Replacement, IEEE Transaction on System, Man and Cybernetic, Vol. 28 (1998) 602-611.
10. Shim J., Scheuermann P., Vingralek R. Proxy Cache Design: Algorithms, Implementation and Performance, IEEE Trans. on Knowledge and Data Engineering, (1999).
11. Squid Internet object cache. http://squid.nlanr.net/Squid.
12. Stenstrom P. A Survey of Cache Coherence Schemes for Multiprocessors. IEEE Computer, (1990) 12-24. 1990.
13. Wills C., Mikhailov M. Towards a better Understanding of Web Resources and Server Responses for Improved Caching, Proc. 8th Intl. World Web Conf., (1999).

Agents Based Collaborative Framework for B2C Business Model and Related Services*

V. Robin Rohit** and D. Sampath***

Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore, India
robinrohit@hotmail.com, dhamusam@lycos.com

Abstract. Agent based software technique is an emerging technology to design and implement software systems with autonomous behaviour and optionally integrate Artificial Intelligence, Knowledge-based system in a seamless manner for specialised activities under distributed computing environments. In this paper we address the issues of incorporating the valued-added services within the B2C model and propose a solution based on the software agents. We present our system, which has been designed for effectively collaborating B2C e-business model with other related service systems like general Customer Services, Customer Retention systems, etc., under the framework of Customer Relationship Management (CRM) using Software Agents. The system has been designed to operate on WWW and has been implemented using JAVA related technologies for the effective deployment over the Internet.

Keywords: Customer Relationship Management (CRM), E-Business, Software Agents, Intelligent Software, Internet, WWW

1. Introduction

Internet was born as a result of the technological advancements in varieties of fields that has enabled virtually all computers to be networked together. Internet technology and World Wide Web (WWW) have changed every facet of human life. The rapid and constant onslaught of new technologies in the IT revolution is wiping out borders in global trade. Through the Internet, businesses can reach markets and potential customers not available before.

Manufacturing industries, banks and financial institutions, and service sectors are embracing the Internet technology for effectively and efficiently managing their process model. Internet based electronic commerce is flourishing particularly in Business-to-Consumer (B2C) world and to some measure in the Business-to-Business (B2B) world as well. E-Business is the term used to refer to the method of doing business by leveraging the Internet technology. However, it is not as simple as the

---

* This work was carried-out while the authors were at Indian Institute of Science, Bangalore

** Department of Computer Science and Engineering, A.K.College of Engineering, Anand Nagar, India

*** Currently at DBS Bank, Singapore

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 126–133, 2001.

statement states, it encompasses various other activities as well. For example, e-commerce, which is a subset of e-business, defined as a system that includes not only those transactions that centred on buying and selling goods and services to directly generate revenue, but also those transactions that support revenue generations. The activities like generation of demands for those goods and services, offering sales support and customer services, or facilitating communications between business partners.

E-Business: Business houses have already made substantial profits through Internet based innovative products and services, which were not existing during pre-Internet era. The emerging Internet related technologies are dislocating existing business processes by new more cost-effective, scalable, net-intrinsic business models [2,3]. In coming years, we can expect the emergence of dramatically new kinds of supply chains, distribution channels, and dynamic markets that use new kinds of intelligent distributed computational process, which we call as agents. The e-commerce model benefited the sellers in terms of large number of customers, reduction in business running cost in terms of overhead, inventory, real estate investment, etc. The profit derived from the above measures has been passed on to the buyers in terms of discount prices and several innovative services to widen the customer base and thereby increasing the volume of business.

In recent time, software agent technology is explored by researchers and practitioners for incorporating intelligence and delegating independent role for software systems in decision making. Agent based software systems have been adapted successfully in various applications like web marts, electronic trading, etc. Last several years of research in e-business has seen many successfully implemented e-business models for Business-Consumer and Business-Business models.

Software Agents: Software agents and multi-agent systems have grown over several years into what is now one of the most active areas of research and development activities in computing. Generally, software Agents are atomic software entities operating through autonomous actions on behalf of the user - machines and humans - without constant human intervention [15]. Rather, we can state that software Agents are computer software systems that are capable of initiating independent and autonomous action in order to satisfy their design objectives. Agents are used as software models in designing system for different kinds of problems, be it a system solution or application domains. While comparing the agents with software design models like objects, we can state that objects encapsulate and have autonomy over their state, where as agents have autonomy over their state and also their behaviour. Thus, for example, there is no notion of method invocation in the agent-oriented world. As agents have control over their behaviour, they must co-operate and negotiate with others in order to satisfy their design goals. In particular, the agent metaphor seems entirely appropriate for domains such as electronic commerce, where the participants must be assumed to be self-interested entities.

The Business-Business model is generally designed for a limited number of interacting business houses for effectively transacting their business processes and thereby achieving efficiency in the overall interaction systems, which benefit all interacting business houses. A large number of successfully implemented e-commerce systems have focused on selling goods and generating income through direct

marketing with millions of potential consumers over the Internet world. However, instead of narrowing the model towards the income generation in a shorter period of return on investment, it is realised by organisations to focus more on the other related services like marketing (to increase customer base), customer retention, customer services etc., to stay in the e-business world. This paper presents a detailed design and implementation of such an integrated approach with software agents as the architectural framework for effectively and autonomously providing better services and knowledge domain components as an integral part of the system.

2. Software Agents and E-Business Models

The advancement in technology, efficiency in the system, and competitiveness in the market resulted in designing novel models to attract customers in any business system. Today's technological development has given more flexibility to customers in selecting widely available options in the market and the world has become a more customer centric, rather than efficiency, which is an assumed factor in today's new economy [11,12]. Business domains like banks, financial institutions, retail and whole sale markets, telecommunication industries, etc., are designing and implementing systems which helps them in analysing their customer behaviour for understanding their need in a better way so as to retain them and grow their base in the business.

There are new tools being developed for mining a large collection of historical data collected over several periods to extract useful knowledge in understanding the interesting patterns present in them. For example, in super market consumer business the data mining tools will extract the pattern of products purchased by consumers and types of products in demand at different intervals and also cluster different types of customers. This kind of knowledge will benefit the organisation to order, and stack them appropriately to serve the customers in a better way at super markets. So, understanding such pattern from on-line transactions is essential to arrive at an appropriate knowledge database and intelligently suggesting the customers on their likely purchasable products from the mart.

In the case of Business-Consumer models, where the products are not just consumable, but needs after sales and services, needs to capture different kinds of information from on-line transactions. Therefore, depending on the types of products transacted in a business, the system should decide the appropriate types of information to be captured, stored and processed. We also need appropriate mechanism for the customers to interact with the system for various kinds of business services. Agent based software systems would be an appropriate selection for implementing a complete system in our example Business-Consumer model of business for implementing various related services of the system, rather than focusing on a limited activities. In the subsequent sections, we explain the architecture of our system and the various software agents designed for such an integrated system.

Apart from the product oriented services for the B-C e-business model, we also have to build systems around the e-business model, which will focus on strategies for increasing the customer base and also retaining the customers for increasing the business volume. Though providing an effective after sales-services for the transacted products would indirectly influence on retaining the customers with the business, one has to innovate new mechanism for retaining the customers in other means and

effectively incorporating with the main system. When we analyse various facets of a business system, one can identify some main activities (wings) of the business system, where one needs to incorporate atomic operations in decision process and at the same time provide effective co-operation among the participating entities in the system. Towards achieving such objectives in the business process, we have proposed the software agents as models for designing an effective system for e-business systems. In what follows, we present the identified components in the e-business system, which are designed as software agents, and also present their design objective and implementation details.

3. Architectural Design and Implementation of the Proposed System

The system designed for an integrated environment has the following agent modules, which operates independently, intelligently and as well as collaborate with other agents. Basically the system has following agent modules and can support any future new modules to be incorporated as part of the overall system.

Interface Agent (I-A): Interface agent is the software module designed for interacting with the new/old customers and collecting essential information about the users and building ***Customers Knowledge System (CKS)*** based on minimum interaction with the customers to collect maximum and relevant information. Considering the future expansion and transformation of the e-business model, we have incorporated the business-to-business interaction for exchanging users common Knowledge, which is not of individual business strategic importance. In the core business software architecture, CKS module is designed to interact with the other marts in the WEB, where the customers are registered users, but prefer to use their identity of other marts to access the currently logged-on mart. Though, this feature incorporated in the design initially seems to be irrational, but we strongly believe that with the globalization of business activities, it is not very far to realise such an interaction among the competing business entities to exchange some vital information about the customers and vendors.

With wider options being available for the customers and vendors, the future business success lies in how uniquely and profitably each business houses spearhead their business for better profitability by new value additions and innovative models rather than locking up their customers and vendors with their business process. It is realised by every business houses that the customers are the key players in the new economy. This new facility of interacting with the peer-mart in the I-A module actually makes it more complex, but versatile so as to transform the pure Business-Consumer business module into Business-Business module in a limited manner in the future period.

Product Selection/Ordering/Processing Agent (PSOP-A): PSOP-A is responsible for providing essential information about the various products offered by the mart and assisting the customers in selecting their needed products. As like any other conventional system, the efficiency of the system depends on how effectively and

elegantly the system assists in the selection process, easiness in the ordering process in terms of mode of payments, payments terms and conditions with authenticity. And among them, building confidence and trust among the participating parties are the key issues in this module. Apart from the above functionality, the agent modules also assists the customers depending on the purchasing characteristics, provides in a transparent mode of the products which are not currently available at the present mart but in other collaborating marts. With such facility, the system designed in not only focusing on the locally available products, but also provides and makes business with the valuable customers on products available on other collaborating marts in a transparent way. Such collaboration among the marts not only multiples the business volumes but also saves the precious time spent on selecting a products by the customers. Therefore, the system is designed in intelligently deciding on the purchasing characteristics of the customers and there by expanding the business horizon beyond a particular mart's boundary. The **(PSOP-A)** agent module continuously updates the ***Product Knowledge System (PKS)*** and also the ***Purchasing Pattern Knowledge System (PPKS),*** which are part of the knowledge bases used in assisting the overall functioning of this module.

Product Service Agent (PS-A): PS-A is the agent module designed to concentrate on the after sales and service functionality of the products purchased through this mart by the customers. As we have perceived in the overall system, it has to be part of the total e-business solution in order to sustain in the market for a longer period. This module apart from the facilitating the service functionality also maintain a knowledge base on various kinds of failures reported with a product and also the suggested remedies used in solving the different types of problems. With such a large collection of databases collected over a period, the system can learn to provide assistance in solving certain kinds of problems reported for a product and also will be used by the manufacturer of such products to solve such a problem in the future production.

Customer Retention Agent (CR-A): Customer retention is a challenging issue faced by many corporations in today's competitive scenario, as the kind of products and services provided by the competing organisations are innovative and pose high competitiveness in the market. The software modules designed for such programme collects some interesting behaviour and interest of the customers in a non-intrusive mechanism. The collected information would be used by the analysing software modules to study different kinds of services and products expected by the customers and will be used later by the decision makers of the organisation in designing new products and services so as to retain the customers with the business. Many interesting reports have been published particularly in the banking and telecommunication industries on this issue and several research papers have appeared in the literature. We have designed the ***CR-A*** module to study the characteristics of the customer interest through discussion forum and on-line suggestion and query mechanism by collecting a vast amount of data and analysing the results. By designing such a system as part of the modules would benefit the organisation in accessing and analysing them and there by taking remedial measures. The architecture of the complete agent based solution is given in Figure 1.

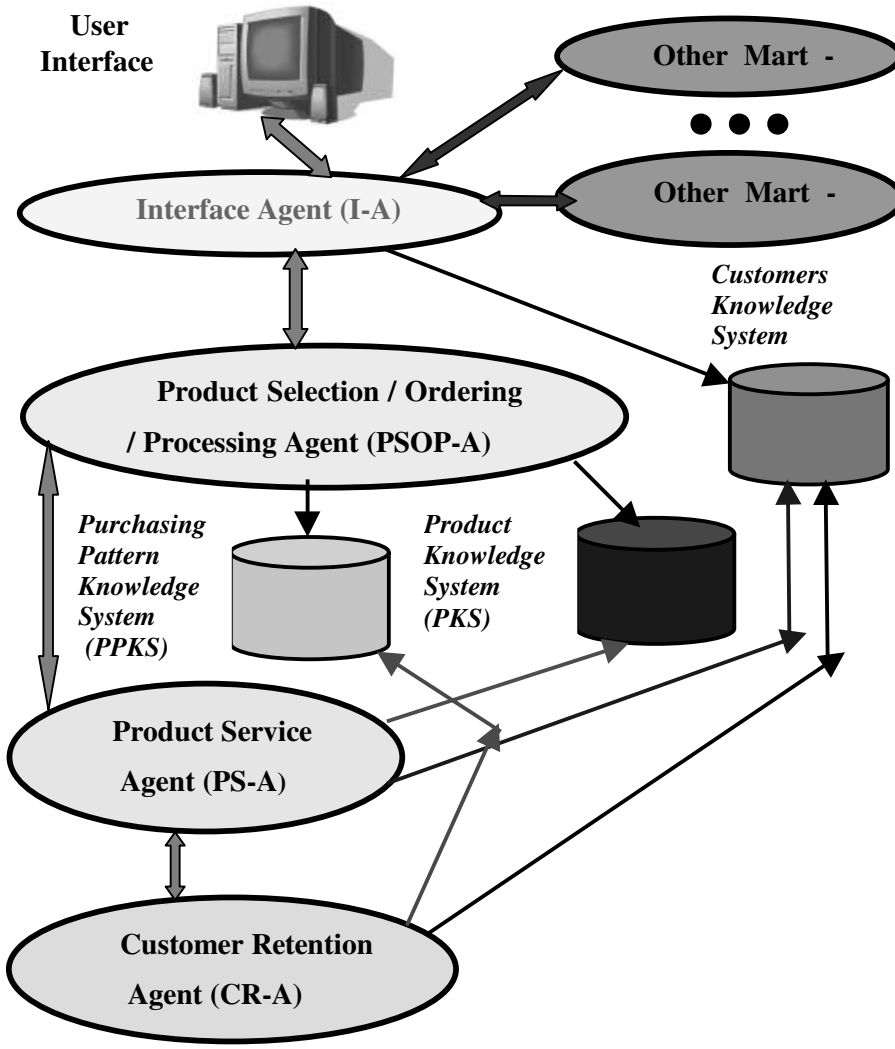


Fig. 1. Architecture of proposed system.

Software Development Environment:

The proposed system has been implemented on Window based system, which supports either Microsoft personal Web Server (PWS) or Internet Information Server (IIS). We have used HTML, Javascript and VBScript for developing the client interface components of the software modules. At the server side of the software components, some modules have been developed using VBScript and ASP pages with Java programs. As like any WEB enabled software systems, one can use either the Netscape Navigator or Microsoft Explorer to access the front-end module of the system. In order to test the system, the back end database used is MS-Access. Since

the database interface components are developed using ODBC components, migration of the database system to other databases systems like ORACLE, etc., becomes simpler as many database developers supports ODBC interface as a standard interface with their system.

4. Conclusion and Remarks

In this paper, we have presented the architecture of the agent based software system for supporting an integrated solution for various systems of the Business-Consumer model. The proposed solution has been successfully implemented using JAVA related technology for WEB deployment. It is shown in the paper that how the agent based software technology can be used to successfully transform the e-business models for embracing the changing business models and transforming the pure Business-Consumer models into Business-Business model. The intelligent agent based software components solves a host of problems related with the customer retention and customer sales services as well as extending the horizon of the business to meet the demands in the market and as well to collaborate effectively with the peer business groups. As part of future expansion, we are incorporating some data mining techniques for on-line transactional data, rather than the applying the data mining techniques on huge collection of historical transaction data. We believe that the post-mortem analysis is one kind of understanding the pattern present in the historical transaction data and one would be more beneficial by analysing the current data for timely actions depending on the patterns discovered in the system.

References:

1. Choi, S.-Y., Whinston, A.B.: The Euture of E-Commerce: Integrate and Customise, Computer, Vol. 32-1. Jan (1999) 133-134
2. Dana Moore, Ed Greengrass: Design Considerations for Agent Systems That Glean the Internet, IEEE Intelligent Systems, March/April (2000) 76-81
3. Froehlich, G.; Liew, W.; Hoover, H.J.: Sorenson, P.G., Application Framework Issues When Evolving Business Applications For Electronic Commerce, Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences (HICSS-32), (1999) 10
4. Gary P. Schneider and James T. Perry: Electronic Commerce, Thomson Learning, Cambridge, (2000)
5. M.Huhns and M.P. Singh, editors.: Readings in Agents. Morgan Kaufmann Publishers: San mateo, CA, (1998)
6. I-JenChiang;Lin,T.Y.: Using Rough Sets To Build-Up Web-Based One To One Customer Services, Proceedings of the 24th Annual International conference on Computer Software and Applications Conference (COMPSAC 2000), (2000) 463-464
7. Jutla, D.; Bodorik, P.; Hajnal, C.; Davis, C.: Making Business Sense of Electronic Commerce, Computer, Vol. 32-3, March (1999) 67-75
8. Marijana Lomic and Zoran Putnik: On Distance Education Courseware, Proceedings Of The 4th Annual SIGCSE/SIGCUE On Innovation And Technology In Computer Science Education, (1999) 194
9. Minar, N.; Gray, M.; Poop, O.; Krikorian, R.; Maes, P.: Hive: Distributed Agents For Networking Things, IEEE Concurrency, Vol. 8.2, April-June (2000) 24 -33

10. Moore, D.: The Changing Face Of The Infosphere, IEEE Internet Computing, Vol. 4.1, (2000) 75-76
11. Ohaegbu,K.;Devgan,S.: Customer Relationship Management In E-Commece: The Call Center Solution**,** Proceedings of the IEEE Southeastcon., (2000) 391-394
12. Reffett,J.L: A Vendor-Customer Relationship To Improve Product Quality Communications, IEEE International Conference on Digital Technology, Vol.2, (1988) 852-856
13. Romaniuk, S.G.: Using Intelligent Agents To Identify Missing And Exploited Children, IEEE Intelligent Systems, Vol. 15.2, March-April (2000) 27-30
14. Weib.G, Editor.: Multi-Agent Systems, The MIT Press: Cambridge, MA, (1999)
15. Wooldridge.M. and N.R. Jennings.: Intelligent Agents: Theory And Practice, The knowledge Engineering Review, Vol. 10-2 (1995) 115-152

Agent-Based Distributed Computing with JMessengers

Moritz Gmelin, Jochen Kreuzinger, Matthias Pfeffer, and Theo Ungerer

Institute of Computer Design and Fault Tolerance,
University of Karlsruhe, D-76128 Karlsruhe, Germany

Abstract. JMessengers is a Java-based mobile agent system particularly designed for distributed computing in heterogeneous computer networks. This paper explains JMessengers programming environment, evaluates JMessengers performance, and compares it to the mobile agent systems MESSENGERS-C and Grasshoppers. The evaluation showed that JMessengers is an optimal compromise between the two systems. It offers the same flexibility as Grasshoppers and reaches almost the execution speed of MESSENGERS-C.

1 Introduction

Numerous approaches exist for high-performance distributed computing in computer networks. Most of these use the message-passing environments PVM or MPI or Software DSM (distributed shared memory) systems like Treadmarks [1], Shasta [2], or Rthreads [3]. To date only the message-passing environments are able to efficiently execute distributed programs on Internet-connected computers, whereas use of Software DSM systems is typically restricted to LAN clusters, in particular to cluster computers that support interprocess activities by special interconnection hardware.

Since the Internet has become the most important structure for computers networks, autonomous Internet agents have appeared as solutions to handle data retrieval and search operations through large numbers of loosely connected entities. The idea of those agents is to allow programs to navigate freely between computers and help the user to manage large quantities of information that would be difficult to handle without the help of such agent systems.

Internet agents come in two different flavors: stationary agents and mobile agents. Only the latter are considered in this paper in more detail. Mobile agent systems are characterized by their ability to autonomously migrate code, data, and execution state of agent incarnations.

Mobile agents define a new paradigm for distributed computing. Typically mobile agents are designed for electronic commerce and therefore hindered for high-performance distributed computing by their inherent security features that slow down execution speed. For distributed computing cooperative agents are applied, thus security issues may be dropped.

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 134–145, 2001.

The idea to combine high-performance computing with distributed machines and mobile software agents to navigate a network, brings us to the MESSENGERS [4] [12] project at the University of California, Irvine. The goal of the MESSENGERS project is to apply mobile agents to utilize the computing power of LAN-connected machines.

In our opinion, the most serious shortcomings of the MESSENGERS-C system are its limitation to 64 machines that must be arranged in a homogeneous LAN cluster. Today, most networks consist of machines using a number of different operating systems. But since all major systems support Java as programming language, we decided to take the very flexible and stable paradigms of MESSENGERS, re-implement the whole system in Java without the limitations of a homogenous network or a maximum of 64 machines, and enhance the original MESSENGERS system by new language and powerful synchronization features motivated by Java.

Our goal was to build an easily scalable platform for high-performance, mobile agent based, distributed computing that can spread programs over Internet, use heterogeneous computer platforms, does not restrict the number of computers used, and enables object-oriented programming. These design goals are met with our Java-based JMessengers system, which is described in the next section. Section 3 explains related mobile agent systems and compares them to the JMessengers system. Section 4 gives some performance evaluations of JMessengers programs and compares JMessengers performance to the MESSENGERS-C and the Grasshoppers systems.

2 JMessengers

The JMessengers system was created as a flexible and fast software platform for mobile agents. JMessengers and all its components are written in Java and are therefore not limited to any special type of machine. A JMessengers network may consist of hundreds of different connected machines. The only requirement for the participating machines is an implementation of Sun's JDK 1.2 standard with RMI. It was very important not to change any of the standards given by Java to be truly system independent.

To reach maximum speed and to have the least overhead, the JMessengers system does not implement any agent verification or authentication. It is therefore not suited for insecure environments, but rather for trusted networks or virtual private networks.

The basic functions of a JMessengers agent—navigation in the network, creation of a new virtual network, and communication between agents—are all implemented in the basic `Agent` class that all custom JMessengers agents must inherit from. JMessengers does also provide the developer with a variety of ways to synchronize agents throughout the network.

2.1 Architecture

The JMessengers system architecture consists of a three-layer design, which is depicted in Fig. 1:

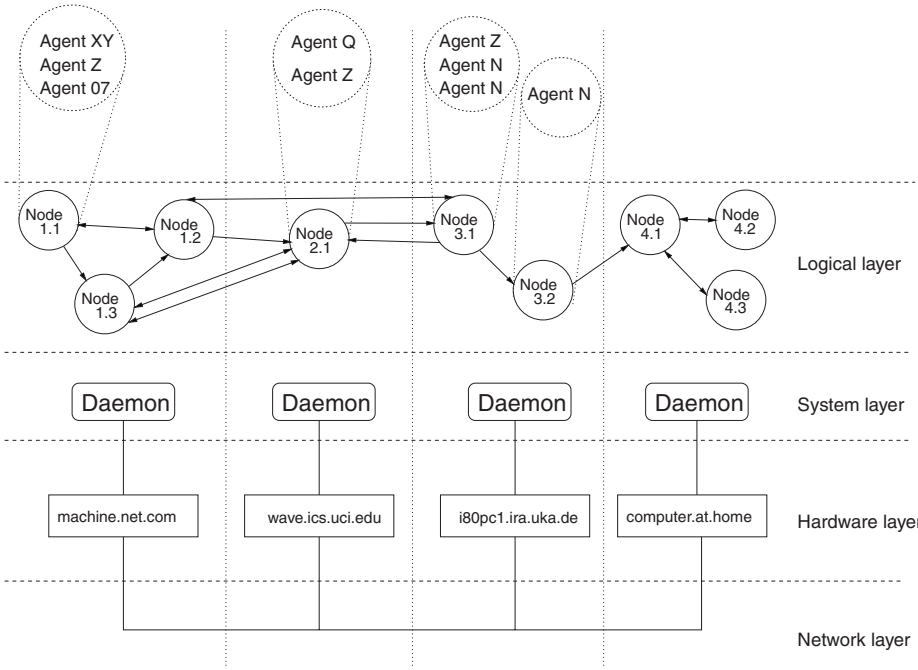


Fig. 1. JMessengers design.

Hardware layer: Machines participating in the network, connected through TCP/IP based LAN or WAN structures.

System layer: A daemon is a program that has to be started on each of the participating machines in order to add that machine to the JMessengers network. All daemons need one master daemon to serve as central gathering point.

Logical layer: On top of the daemons, a JMessengers system uses virtual nodes and links to overcome the limitations of the underlying physical layer. Each JMessengers daemon can host an unlimited number of virtual nodes, connected to each other and to nodes on other daemons through virtual links. Those nodes are used to host the agents.

2.2 Usage of JMessengers

To build a JMessengers network, the user must first start one master daemon on any of the machines in the network. This master daemon serves as a central gathering point to all other daemons. It also hosts the initial node of the JMessengers system. Every other daemon that the user starts needs a reference to the master daemon in the system, which must be provided at startup time by the user. The master daemon then keeps track of every registered daemon and is responsible for the following tasks:

User interface: On startup of a master daemon, a user interface is shown on the machine's desktop. Through this interface, the user can

- survey the virtual network,
- cleanup the created network (delete all virtual nodes except the ever existing initial node),
- shut down the whole system and stop all working daemons,
- inject a new agent onto the initial node of the system.

Daemon registration: The master daemon keeps track of all other daemons in the system and receives new daemons in a network, even at runtime.

Code distribution: If the code of an agent (Java bytecode) does not exist on one of the daemons (nodes) that an agent hops to, the receiving daemon asks the master daemon to provide him with that code.

Node balancing: Upon the creation of a new virtual node, the master daemon is responsible for equally distributing the nodes on all available daemons (machines). Besides this way of automatic distribution, it is also possible to choose a particular daemon on which the node will be created.

2.3 Functionality

The JMessengers agent system and its class framework provide methods for most applications. All agents must be subclasses of the `Agent` class. This class already implements methods for:

Network creation: Upon the start of a new JMessengers network, the master daemon creates one initial node. This node is the starting point of each agent in the system. From there the agent can use the `createLink()` method to create new links and eventually nodes. New nodes are automatically created on the next available machine if the link is not directed to an already existing node. Each link is identified with a (non unique) name, a weight and a direction. That means that links can be created as being uni- or bi-directional. The numerical weight and alphabetical name of a link are used to identify it.

Network navigation: After a network is established, an agent can travel along links using the `hopLink(`**Link** `linkObject,` **String** `nextMethod)` method. Traveling along a link means:

1. The agent is serialized (the content of its global variables are packed in a byte-stream).
2. The serialized object is copied to the destination node.
3. Deserializing the object to restore the state of that object as it was on the originating node.
4. Starting a new thread for the agent and resume execution with the method that the user did specify in the `hopLink()` command.

There is also a `hopNode(`**Node** `nodeObject,` **String** `nextMethod)` method to bypass links and directly hop to any existing node. Both ways to travel in the

network use *soft* migration. This means that the agent still lives after a hop on its original node and can continue execution there. The agent is only replicated to its destination node and started as a new thread at a point specified by the user.

This is different from most other agent systems. It also differs from MESSENGERS that implements traveling as *hard* migration, meaning that the execution of the agent is stopped on the originating node and only continued on the destination at exactly the point where it stopped before the migration.

Inter-agent communication: The communication between agents is done through shared data in node variables. Therefore, an agent can use the `putVar(String name, Object obj)` methods of its `Agent` base class to store data on its actual node. Any other agent can then jump to this node and retrieve the stored data by its name using the `getVar(String name)` function.

Inter-agent synchronization: In a system, where multiple autonomous agents coexist and are supposed to cooperate, synchronization is indispensable. Therefore, JMessengers is enhanced by the `MSync` class. Objects of that class can be shared among agents and even among agents on different machines. The `MSync` class provides the following synchronization methods between agents:

- **Counter synchronization:** Every call to the `getCounter()` method of an `MSync` object results in an atomic counter incrementation. This can be used to split work between an unpredictable number of working agents.
- **Barrier synchronization:** The user can specify a number of agents that must have reached the barrier by calling the `limitSync()` before all those agents continue execution.
- **Client/server synchronization:** A client that calls the `syncClient()` method of an `MSync` object is suspended until any server calls the `syncServer()` method and vice versa.
- **Event synchronization:** An agent that calls the `waitEvent (String name)` method is suspended until another agent calls the `signalEvent (String name)` method on the same `MSync` object with an identical name as argument.

All those synchronization methods enable the cooperation of large numbers of agents in JMessengers networks. There can also be multiple `MSync` objects active at the same time in the network to independently synchronize different groups of agents. The synchronization methods enhance MESSENGERS' synchronization abilities and are easily implemented on top of Java.

2.4 Example for a Simple Agent Program

The programming of a JMessengers agent is demonstrated by the simple agent program in table 1 that creates a new link `Link_1` from the initial node to a new node `Node_1`, jumps over this link, jumps back and finishes. At each node, the agent prints a message to show the user where it is.

Table 1. Example JMessengers program to jump over nodes

```
public class ExampleAgent extends Agent
{
  public void start(String args[])
  {
    //Print greeting message
    System.out.println ("Agent started at initial node");

    //create new link
    Link l = createLink ("Node_1", "Link_1", 0, true);

    //hop over new link and then resume at the "jumpBack" method
    hopLink (l, "jumpBack");
  }

  public void jumpBack()
  {
    //Print message
    System.out.println ("Just arrived at node " + getName() +
                        " at host " + getHost()");
    System.out.println ("Will be jumping back home");

    //get the link back
    Vector v = getLinks("Link_1");
    Link l = (Link)v.elementAt(0);

    //Jump
    hopLink(l, "end");
  }

  public void end()
  {
    System.out.println ("Back home. Now finishing");

    //End. The Agent is now garbage collected.
  }
}
```

3 Related Work

JMessengers can be viewed as a Java version of the MESSENGERS system with enhancements in synchronization, programmability, and heterogeneous platform usability. The roots of JMessengers may be traced back to the WAVE system [5] which itself is a predecessor of MESSENGERS. The MESSENGERS system was designed as a mobile agent system for high performance computing on a local network. Its first implementation [6] was based on a C-like script language, which was interpreted by the daemons. Due to the interpretation, the system was not very fast, but the principles of mobile agent computing could be investigated. The second implementation takes care of the main drawback, the performance, leading to the compilation-based MESSENGERS-C system [7,8]. This version works like a precompiler that translates the slightly modified C-code of MESSENGERS-C into pure C-code, which can be compiled by a standard compiler. The MESSENGERS-C system proved that mobile agents programs might be as efficient as PVM programs. The two main disadvantages of MESSENGERS-C are: first, all computers in the network need to be binary compatible (all work with Solaris *or* all with Linux) and second, the maximum

number of connected computers working on one problem is restricted to 64. Both drawbacks were solved with JMessengers due to Java as the programming and implementation language and the use of RMI for scaling the system up to more than 64 computers.

Two commercially available and comparable mobile agent systems are Grasshoppers from IKV GmbH [9] and IBM's Aglet system [10]. Both systems are Java-based mobile agent systems for distributed computing, like JMessengers. However, a main focus of these systems lays on the security of and for the agents, which is in contrast to JMessengers. The Aglet system verifies the execution of incoming agents by a security manager over authentication and controls the resources used by the agent. There is no logical network layer over the physical network, so each host is equal to a node. The communication between the agents is done via *proxy objects*, which are made available through Aglet methods.

Grasshoppers is based on the MASIF (**M**obile **A**gent **S**ystem **I**nteroperability **F**acility) standard [11], which provides a uniform interface for agent platforms. Due to the underlying CORBA layer, it is possible to integrate traditional client/server models into mobile agent systems. For communication and navigation it also uses RMI proxy objects, but also CORBA, IIOP or pure socket connections. Optionally the data can be encoded through SSL and an authentication of incoming agents is also present. Grasshoppers allows on system or user level to save the state of agents and reload it after a system crash or on demand.

4 Performance

Three different types of performance measurements were done: communication tests, calculation tests and the combination of both. We compared JMessengers to MESSENGERS-C and Grasshoppers. The comparison between MESSENGERS-C and JMessengers is the fairest one, because both systems have been developed with the same purposes. The comparison between JMessengers and Grasshoppers must take into account that Grasshoppers offers more security and supports more protocols.

The first test (see Fig. 2) compares the communication time of MESSENGERS-C and JMessengers using two Sun SparcStation 5 workstations connected over a 10 Mbit Ethernet. An agent jumps between two nodes and carries some data with itself. The nodes can be on the same (1 host) or on different computers (2 hosts). Figure 2 shows the time for 1000 hops in dependence of the transported amount of data.

We see, that the MESSENGERS-C system needs almost no time for jumps between two nodes located on the same computer. In contrast to that, JMessengers has to serialize the agent even though the destination is located on the same daemon. This consumes a significant amount of time. JMessengers jumps between computers are slowed down 20ms each by synchronization packets that are sent by RMI before the actual jump. Those synchronization packets are delayed by the operating system, that waits for 20ms before sending a non-full

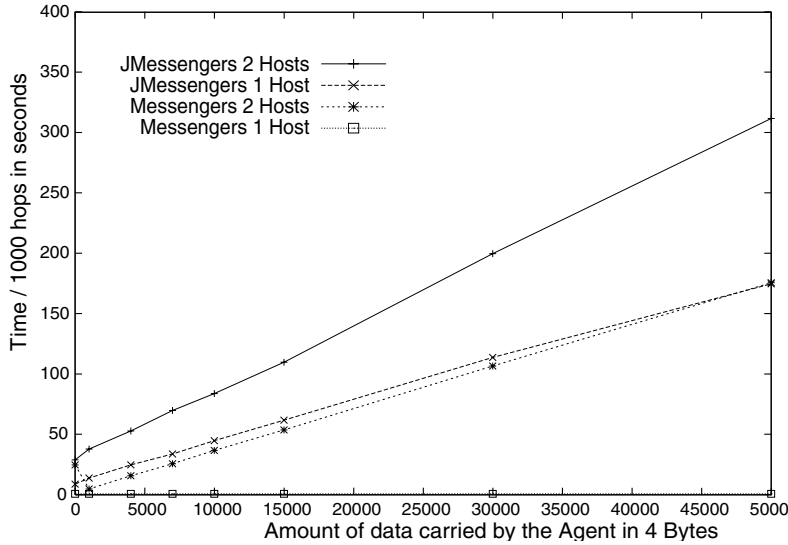


Fig. 2. Communication times of MESSENGERS-C and JMessengers.

TCP-stack. The same effect is visible for MESSENGERS-C when sending Agents with very little or no data. So jumps between separate machines in JMessengers take as long as in MESSENGERS-C plus the time needed for serialization plus 20ms.

The next test compares JMessengers to the Grasshoppers system. Grasshoppers is a commercial agent system, however we used a free test version. It differs from the full version by the lack of security mechanisms. This lack in the free version has no influence on the comparison of the two systems. This communication test is almost the same as the previous communication test with MESSENGERS-C. We reduced the number of jumps to 100 to keep the runtime of the tests in a reasonable range.

Figure 3 shows that Grasshoppers has a large overhead for jumps. The time for jumps within one machine is almost constant and only slightly depends on the size of the transported data. That is possible, because Grasshoppers does not copy the agent. When jumping between computers, both systems have to serialize the objects. The network transportation times of the objects are the same in both systems, but JMessengers performs better than Grasshoppers. The advantages of Grasshoppers are security and support of different protocols.

The previous tests did not vary the ratio between communication and computation. The variation of this ratio is the subject of the following test. Therefore, we kept the number of agents constant. The workload for this test is the calculation of a fractal image (Mandelbrot set of 2048x1600 pixels). The image is divided into equal sized partitions, which are computed by a constant number of agents. We vary the number of partitions the image is divided into. This test is done on four Sun SparcStation 5 workstations.

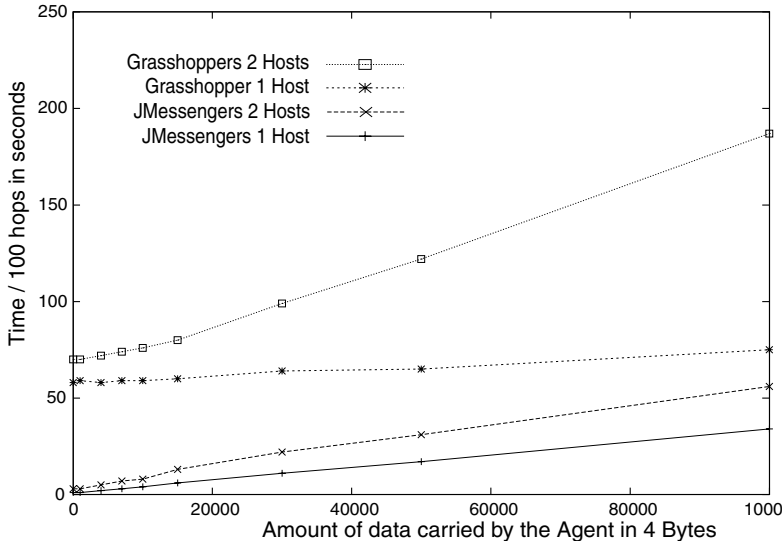


Fig. 3. Communication times of JMessengers and Grasshoppers.

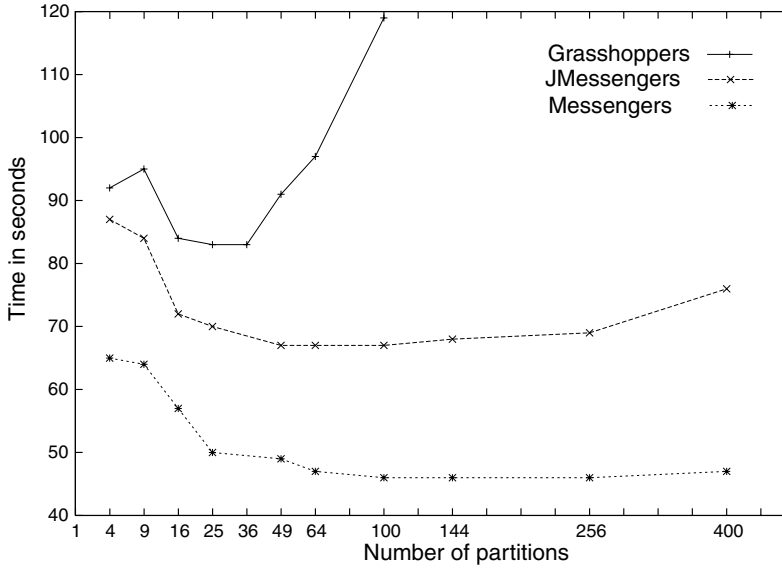


Fig. 4. Variation of the communication-computation ratio.

Figure 4 shows that the Grasshoppers system needs a lot of time for execution. At the beginning it takes advantage of the finer granularity. Later, the overhead emerging from the jumps exceeds the advantage of the more regular distribution of the jobs. Both, JMessengers and MESSENGERS-C, do jump faster. With an increasing number of partitions they can use the calculation time

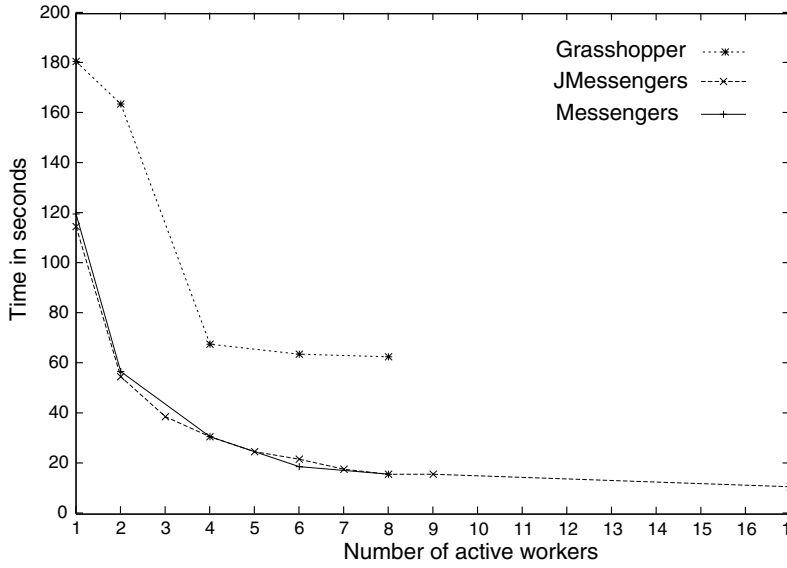


Fig. 5. Scaling of the systems.

of all computers more equally. More computers can be used in a profitable way. The JMessengers agents jump slower than the MESSENGERS agents. The time difference is almost constant at 20 seconds. At a number of 800 partitions this difference rises up to 30 seconds.

The final test evaluates how the different systems scale with the number of agents. Again the fractal image calculation has been used, however, for these tests we used more and faster machines, in particular 10 UltraSparc 10 and 10 UltraSparc 5 workstations, and a larger image (3000x2000 pixels). We increase the number of working nodes and agents to have more workers split the task.

Figure 5 shows almost no difference between JMessengers and MESSENGERS-C. The execution times of the Grasshoppers system are much higher. A reason for that could be the inadequate speed for jumps. We can see, that for an amount of more than eight agents (machines) the performance gain is getting small, probably, because the problem size is just not big enough.

5 Conclusion

We introduced and evaluated the performance of the Java-based JMessengers system, which is based on the MESSENGERS system of the University of California, Irvine. The compilation-based MESSENGERS-C system already showed that mobile agent programs might be as efficient as PVM programs. However, only LAN clusters of homogeneous computers with up to 64 computers are supported by the MESSENGERS-C system. These drawbacks are solved with JMessengers due to Java as the programming and implementation language and the

use of RMI for scaling the system up to more than 64 computers. The cost, which must be paid, is a slight performance degradation. Java allowed to provide the JMessengers system with powerful synchronization constructs that are not present in MESSENGERS

The three platforms compared to each other in our performance tests can be ranked as follows: MESSENGERS-C is especially suitable for applications on computers with restricted resources, it doesn't support large networks due to the limitation to 64 computers. MESSENGERS-C should be used when there is no appropriate Java environment available.

Grasshoppers is suitable for distributed networks because of its security mechanisms and its support of different network protocols. The agent jumps take a long time in this system. Grasshoppers is a commercial product. It offers a graphical user interface, which makes the monitoring of the network easy. The Java-based system allows the integration of different computers in a heterogeneous network. Programming is eased by the object orientation and the powerful class library of the base system.

JMessengers is an optimal compromise between the two systems in respect to flexibility and execution speed.

Acknowledgement

We would like to thank Lubomir Bic, Michael Dillencourt, and Eugene Gendelman of the MESSENGERS research group at University of California, Irvine, for their assistance in developing the JMessengers system.

References

1. Pete Keleher and Alan L. Cox and Willy Zwaenepoel *Lazy Release Consistency for Software Distributed Shared Memory.* Proc. of the 19th Annual Int. Symp. on Computer Architecture (ISCA'92), 13-21, 1992.
2. Daniel J. Scales and Kourosh Gharachorloo and Chandramohan A. Thekkath *Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory.* Proc. of the 7th Symp. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), 174-185, 1996.
3. M. Zahn, T. Ungerer *Home-based Release Consistency in Object-based Software DSM Systems.* Eighteenth IASTED International Conference Applied Informatics, Innsbruck, Austria, 14-17, February 2000.
4. http://www.ics.uci.edu/˜ bic/messengers
5. P. S. Sypaty *WAVE-1: A new ideology of parallel and distributed processing on graphs and networks.* Future Generation Computer Systems, 4:1–14, 1988.
6. Lubomir Bic, Munehiro Fukuda, Michael Dillencourt *Distributed Computing using Autonomous Objects.* IEEE COMPUTER, August 1996
7. Christian Wicke, Lubomir Bic, Michael Dillencourt, Munehiro Fukuda *Automatic State Capture of Self-Migrating Computations in MESSENGERS.* Second Int'l Workshop on Mobile Agents 98 (MA'98), Stuttgart, Germany, September 1998

8. Christian Wicke, Theo Ungerer, Lubomir Bic, Michael Dillencourt *Mobile Agenten und Cluster Computing am Beispiel des Messengers-Systems.* Proc. of the 2nd Cluster-Computing Workshop, University of Karlsruhe, Chemnitzer Informatik-Berichte CSR-99-02, 145-154, March 1999.
9. http://www.ikv.de/products/grasshopper
10. http://www.tri.ibm.co.jp/aglets
11. OMG, Object Management Group http://www.omg.org
12. Christian Wicke, Lubomir Bic, Michael B. Dillencourt and Munehiro Fukuda. *Automatic State Capture of Self-Migrating Computations in Messengers.* http://www.ics.uci.edu/˜ bic/messengers/papers/MA98.ps

Agent-Based Wave Computation: Towards Controlling the Resource Demand

Armin R. Mikler and Vivek S. Chokhani

Department of Computer Science,
University of North Texas,
Denton, Texas 76203, USA
{mikler,chokhani}@cs.unt.edu

Abstract. In recent years, the mobile agent paradigm has received significant consideration in the context of large complex decentralized systems. Tasks such as system monitoring, load balancing and resource management have been successfully mapped onto this paradigm. Although the significance of the agents' resource demand has been recognized and discussed, few mechanisms have been developed to control the demand as a function of resource availability. In this paper, we propose an agent based computing model that incorporates the concepts of wave computation. In particular, mechanisms to dynamically adapt the agent population are investigated. Related issues, such as wave propagation and termination are discussed. Further, we present the results of a variety of simulation experiments used to investigate the properties of the proposed algorithms.

1 Introduction

The mobile agent paradigm has attracted attention from many fields of computer science. The appeal of mobile agents is quite alluring - mobile agents roaming the Internet could search for information, meet and interact with other agents that roam the network or remain bound to a particular machine. Agents are being used or proposed for an increasingly wide variety of applications, ranging from comparatively small systems to large, open, complex real time systems. They could also be very effectively used in a large decentralized autonomous cooperative system, i.e. a system that provides fundamental services for integrating high-level services in a large distributed environment with local autonomy for individual platforms and processes. In these types of applications, knowledge of node based parameters is often essential to make rational decisions. Load balancing and network routing are typical examples of such applications. The decision mechanism of a load balancing system may require explicit knowledge of the current load at every node. To efficiently route packets through a large communication network, the constituent network nodes may require topology information for generating the routing maps or routing tables [3].

Many network management systems today are designed and configured according to a centralized network paradigm. Here, a single system collects, aggregates, and processes all data by itself. Simple polling is still the most commonly

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 146–158, 2001.

used method. Even a small network may have many heterogeneous components, which makes common network management tasks very difficult to execute. As computer networks continue to grow in size and complexity it becomes infeasible to prefabricate systems capable of correctly responding to all possible scenarios. In many cases, it is inappropriate to manage networks in centralized fashion. Network management by delegation provides an alternative to many such centralized network management practices [6].

The mobile agent paradigm represents a powerful approach that may yield a very elegant solution for achieving this delegation of tasks. Rather than providing services to a user at the application level, agents are considered as an integral part of system level software and perform tasks that are considered central to the distributed system. Computation based on the mobile agent paradigm is inherently distributed and decentralized. Hence, it can be viewed as wave or diffusing computation [1,4]. A diffusing computation is started when a node or host sends a message to one or more of the nodes in the network, which after receiving the first message forward it to their own successors.

In this paper we discuss the implementation of a computation wave that diffuses through the network as emulated by mobile agents, traversing the network to complete a computational task. Mobile agents can traverse sequentially through a set of nodes spawning child agents that perform the computation in parallel. However, the degree of parallelism must be controlled as a function of the available network resources. In what follows we propose a mechanism for controlling the resource demands of the agent based wave computation by restricting the population of constituent agents.

1.1 Motivation

Recent research effort has focussed on various issues related to agent mobility and agent collaboration in multi-agent systems [2]. Research results confirm that multi-agent systems represent an effective paradigm for large decentralized systems. However, thus far, the agent resource demand has not been taken into consideration. In order to avoid compromising the utility of the distributed system, it is necessary to monitor the allocation of resources to agents. In other words the size of the constituent agent population needs to be controlled.

Parallel Computation Agents are independent computational entities hence, multiple agents can be utilized concurrently to exploit the inherent parallelism of the agent approach. In general, multi-agent systems can offer greater utility as compared to single-agent systems. For any parallel processing approach we need to consider the time optimality, and most importantly the cost optimality. An agent executing on every node in the distributed environment would be an ideal situation however, system resources need to be carefully managed. In fact we may trade-of computational optimality with resource cost, such that for a given application the optimal number of agents may be significantly less than the number of nodes in the system. Given a large number of system parameters

involved, this optimum would be difficult to derive a priori. In addition, the optimal size of the agent population may dynamically change as a function of the progression of the computation. Strategy proposed in this paper introduces parallelism as required, by allowing agents to clone themselves, or merge with other agents residing on the same node. A computation in our system starts with a single agent, and expands through cloning and then contracts as the agents merge and assimilate each other. The agent computation wave is said to have terminated when the population contracts to a single agent.

Autonomous Computation We are considering an infrastructure involving a society of agents. All the interactions among agents or between agents and nodes should be inherently decentralized. In other words, the population control should be an intrinsic or inherent characteristic of mobile agents and should not be imposed by a global third party. Agents must make autonomous decisions about when and how to populate the distributed environment. Investigation of the requirements and the architecture for the mobile agent infrastructure has been the focus of research in the agent research community [9].

Carefully chosen decision policies are frequently hard-coded in every agent. Those policies that appear profitable from the perspective of each individual agent are not necessarily efficient for the system as a whole when pursued by multiple, interacting agents. Hence, autonomous decisions, though made locally by individual agents, should satisfy the criteria of cost, time and resource optimality globally.

2 Network Monitoring as Wave Computation

Many network management problems can be generalized as node-based parameter acquisition and processing. The agents migrate among the nodes in the network in self-determined manner ensuring that they would acquire complete topological knowledge [10]. We emphasize the self-determined manner of traversal to ensure the decentralization of decision power in the network. The information at a node can be static or dynamic. If the node has static information, we can gather all information once and would need no further actions. However, all network information is dynamic, such as routing information, resource availability and other node data. Mobile agents have several advantages in distributed information-retrieval applications. By migrating to an information resource, an agent can invoke resource operations locally, thereby eliminating the network transfer of intermediate data [8]. An agent can dynamically choose different migration strategies depending on its task and the current network conditions. For experimentation, we have simplified the agent's task by making the agent gather local data at a node in a static network topology. This task is implemented by agent based wave computation. The simulation begins with a single agent visiting a node. When the agent finds new information, it creates an identical clone. Thus, the agent population increases as in a wave computation. Once a sufficient number of agents are present in the system, most of them would not encounter

new information and hence will merge with other agents when they visit the same node. This will lead to a reduction in the agent population and thus the termination of the wave resulting in a single agent having complete knowledge of the network.

2.1 Agent Mobility

By expanding the agent population for a brief period of time, the monitoring task implicitly increased the amount of parallelism, thereby accelerating the progress of knowledge acquisition. As the rate at which agents acquire network state information starts to decrease (i.e., the agents do not acquire new information), the agent population will autonomously decrease. Nevertheless, rather than contracting the population size to a single agent, as it would be reasonable for a network with static resources, the constituent agent population could potentially depend on the dynamics of the network to be monitored and among other things, the availability of resources.

The various phases of an agents life cycle are described in Figure 1. The agent initially enters a node and signals its arrival at the node. Thereafter, it checks if there is another agent present in the node. If there is another agent, the incoming agent waits to merge with the agent that arrived prior to itself. If there is no other agent in the node, the agent executes at the node. After completion of execution and before departing from the node, the agent checks again for other agents that might have arrived at the node while it was executing. In case there were new arrivals, the agents merges with the waiting agents. The agent then schedules its departure from the node. Specifics of agent based monitoring are briefly stated below.

1. **Input Phase**
   When an agent arrives at a node, it remains in the input phase for a specific amount of time T_i. During this time, it checks for other agents already at the node with whom he can merge. Hence, T_i is an important parameter that controls the merging activity. If no other agents are found, the agent moves to the next phase after signaling its presence at the node.
2. **Knowledge Acquisition Phase**
   In this phase the agent actively acquires information that is available at the node. In some cases it may be possible to pre-process (filter) the information before storing it in the agent's knowledge base. For efficiency reasons it is imperative to consider different ways of representing the information. This, however depends on the specific task and is beyond the scope of this paper. Time-stamping of information might be necessary, to make merging decisions in the input phase based on an approximation of the network dynamics.
3. **Cloning Phase**
   Once the agent has completed its task at the node, it moves to the cloning phase. In this phase, agents may be cloned (duplicated),thereby increasing the size of the agent population. The decision whether or not to clone is controlled by a decision function that may depend on several system related

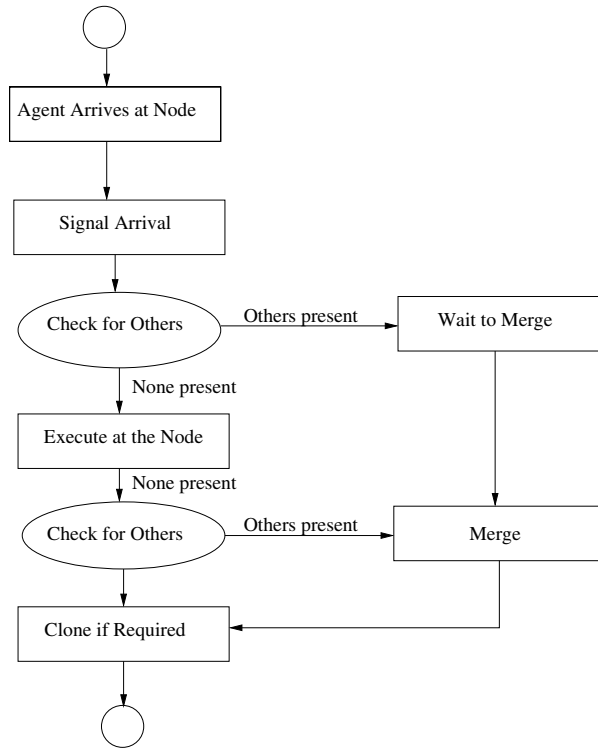


Fig. 1. Agent state diagram.

parameters, e.g., node connectivity, specific agent related variables, or state of the agents knowledge base.

4. **Migration Phase**
   This phase encompasses all the actions necessary for an agent to leave the current node and to migrate to other nodes. Before the agent departs from the node, it checks for other agents that might have arrived during its execution for merging possibilities. If no agent has arrived, it continues with its departure scheduling. Depending on the specific task, the new node may be chosen as a function of the topology or dictated by the agent's itinerary. For network monitoring tasks, the decision of selecting the next hop for the agent maybe supported by the underlying routing mechanisms.

2.2 Wave Computation

Agent mobility, as introduced in Section 2.1 can be the basis of wave computation as discussed in [1,5]. We consider an agent as the manifestation of a message, for information retrieval tasks. An agent is initiated at some network node. At successive nodes during the traversal, it multiplies giving rise to an agent wave

progressing through the network. Any computation is a collection of events, partially ordered by the causal precedence relation.

A computation qualifies as wave computation if it has at least a *decide*, *dependence* and a *termination* event [4]. A *decide* event is a special internal event preceded by a *dependence* event, i.e., for any wave computation, there exists at least one event depending on another event that precedes it, based on which a decision is made about the continuation of the computation. The *termination* event ensures that the computation terminates.

The progression of agent population maps very closely to this definition. Because of the decentralized nature of the approach, there is no explicit communication between two agents. Hence every agent determines whether it is the last agent remaining in the system. The last agent, after accomplishing its goal, declares the wave terminated. This declaration can be viewed as the *decide* event for the agent based wave computation. Every decide event is preceded by the information retrieval process implemented by the agent population. Cloning is implemented as a function of knowledge gained. Agent based conscientious traversal augmented by agent collaboration ensures complete topology traversal [2]. Once an agent acquires complete knowledge of topology, the cloning is terminated. The merge is independent of the task and would be executed in the event of agents meeting. Construction of merging and cloning assures termination of the agent wave.

3 Cloning, Merging and Termination Detection

We have simulated an agent wave based network monitoring system, to study the control parameters for an efficient wave algorithm. As indicated earlier, the agent wave is controlled by two operators, *clone* and *merge*. The agent uses its knowledge base to decide upon the operation to be executed. The specifications of such a strategy are discussed below.

3.1 Cloning

When an agent moves to a new node, it may acquire new knowledge. If the agent acquires considerable amount of new information, then cloning is a sensible decision. Cloning increases number of agents employed for the same task, consequently increasing the parallelism. However, cloning must be controlled so as to avoid infinite and unnecessary duplication of agents. Infinite cloning clearly causes explosion of agent population. Control mechanisms to avoid this are discussed below.

1. *Deterministic Cloning*
   In *deterministic cloning*, the agent clones whenever new information is acquired.
2. *Threshold Cloning*
   In *threshold cloning*, the agent clones when knowledge greater than a certain

threshold value is acquired in a single time-step. This threshold may be a fixed value or could be dynamically computed as a function of cloning parameters. Many such variations are possible. We may pre-set the threshold as a fixed percentage of knowledge gained per time-step or the percentage could be varied as a function of system dynamics.

3. *Probabilistic Cloning*
   In *probabilistic cloning*, the agent clones probabilistically, i.e., at the beginning of the wave, agent would have little or no knowledge about the topology, so it makes sense to clone with high probability initially. As the wave advances and the agents move closer to their goal, they may reduce the probability of cloning. In this approach it is mandatory that the agent has some measure of success. In a highly decentralized and distributed approach this is very difficult to accomplish, as success is due to collaboration of several independently executing computational entities.

In this paper, *deterministic cloning* was explored. A probabilistic version of *deterministic cloning* was implemented with fixed probabilities rather than dynamically changing probabilities as suggested above.

Unnecessary cloning may lead to the generation of a large number of agents that only acquire restricted knowledge over their lifetime. This in turn may result in slow convergence of the knowledge base and excessive resource demand. Hence, the termination of the wave computation may be delayed drastically. This scenario arises if the agent itineraries are not properly diversified. If a cloned agent always follows its parent agent, it would never contribute to the global knowledge acquisition task, yet consume system resources. This implies that after cloning, parent and child agents should follow *diverse* itineraries. This will result then in maximum collaboration with a minimum number of agents. In general, cost optimality greatly depends on diversification.

Cloning produces new agents that are equivalent to their parents in all respects. The new agent has the same decision power as its parent, and it may clone itself further. As per the current mechanism, we start with one agent, which could be pictured residing at the root of the agent family tree, where every new agent created is a child.

3.2 Merging

When two agents meet at a node, one of them may be eliminated to reduce redundancy. Knowledge exclusively gathered by that agent would be lost due to elimination. Hence, merging should always be preceded by knowledge exchange. The remaining agent is a representative of two merged agents, hence further traversal should be conscientious with respect to both knowledge bases, i.e., the resulting agent takes the history of both agents that have merged into consideration when making decisions about cloning, merging and migration.

Population explosion is avoided by executing the merge operation. As agents near the end of their traversal, the cloning reduces and ultimately stops. By the virtue of design, agents cannot reside on an edge between two nodes. So if agents

are always made to merge in the event of agents meeting, the number of nodes in the network represents a bound on the number of agents.

Just like different heuristics for cloning, different merging strategies may be possible:

1. *Deterministic Merging*
   In *deterministic merging*, agents always merge when they arrive at a node in the same time-step. The knowledge bases are merged and the new agent follows a path according to the knowledge gained from the merged agents.
2. *Heuristic Merging*
   In *heuristic merging*, the agents merge when their corresponding knowledge bases overlap beyond a certain similarity threshold. Here, it is assumed that agents with diverse knowledge bases are likely to continue to move in diverse areas of the network and hence, it is better not to merge them. On the other hand overlap of information indicates overlap in the paths of the agents and hence it is better to merge them.

This paper explores *deterministic merging*, where agents merge whenever they arrive at a node at the same time, as indicated in Figure 1 and explained in Section 2.1.

3.3 Wave Termination and Termination Detection

During the initial stages, it is expected that there will be an increase in the agent population and a reduction later on until the agent population reduces to one. During the decline phase, it is quite likely that due to merging one agent acquires all knowledge about the network. It is in best interest to terminate the simulation at this point. However, since it might not be the last agent, it will lead to further merge operations until there is a single agent with all knowledge. This causes a gradual population reduction which must be reduced in order minimize unnecessary utilization of the resources. This behavior may be due to the fact that there are few agents in the network which are moving in a manner such that they do not meet and hence will not merge. To take care of this problem, we need to introduce the following conditions.

1. *Disabling Cloning*
   If the agents do not gain new information for some number of visitations, they decide not to clone any further, even if they get new information from the nodes. This inhibition of cloning ability ensures that there is no further increase in agent population.
2. *Disabling Movement*
   After reducing the cloning probability to zero, the agent will continue to traverse the network. If no new information is acquired for few other visitations, the agent will enter a state that forces the termination of migration when reaching the node at which the wave has originated. This strategy will be useful if a small number of agents are unable to merge due to the random nature of migration.

Every wave computation by definition terminates. An interesting issue is termination detection. As indicated in Section 2.2, the *decide* event in agent wave computation is the declaration of termination by the last agent. Hence it is necessary that the last agent is able to determine that it is indeed the *last* agent. Termination detection is based on a modified *credit share* scheme [7]. We are deploying binary cloning, i.e., cloning always produces only one new agent. Thus the outcome of cloning is two identical agents. The merge operation is also binary, as it works on two merging agents resulting in a single agent. This binary nature of the *clone* and the *merge* operations allow us to simplify the *credit share* scheme and apply it to agent based wave computation. The binary *credit share* scheme involves two parameters, namely $Numerator(N)$ and $Denominator_{log}(D)$ (log of denominator to base 2). The *credit share* of an agent is computed as $N/2^D$.

The Binary Credit Share Algorithm The first agent starts with a *credit share* of 1, i.e., it has $N = 1$ and $D = 0$. Upon cloning a new agent is created. The N of the original agent is copied to the N of the clone. The D of the original agent is incremented by 1 and copied to the clones D. Thus, both agents now have identical *credit share*.

When merging two agents are being consolidated. For example, consider $agent1$ merges with $agent2$. Further assume $agent1$ has an equal or higher D value as compared to $agent2$, i.e. agents from different generations may merge. D of the resulting agent is computed as the maximum of the D of the two agents. N is calculated as, $N = N_1 + (2^l * N_2)$, where N_1 belongs to $agent1$, N_2 belongs to $agent2$ and l is the difference in the D values of the two merging agents.

We are imposing the following invariant A: $\sum_{\forall Agent\ a_i} N/2^D = 1$

Proposition 1. *Invariant A will be maintained throughout the proposed wave computation.*

Initially, the first agent maintains $N = 1$ and $D = 0$. Upon cloning the *credit share* is identically distributed among the two agents and hence the invariant is still satisfied. During the merge operation the *credit shares* of the two agents are added and inherited by the resulting agent. Hence, the amount of *credit share* in the system will not be altered by a merge operation. By induction, the invariant is always satisfied during the entire wave computation.

Proposition 2. *Upon termination of computation the credit share of the last agent is*
$N/2^D = 1$

From conjecture 1, invariant A is satisfied throughout the computation. In particular, for the last two agents in the wave,

$$\textit{Credit share of } agent1 + \textit{Credit share of } agent2 = 1.$$

Hence after the merge operation, the last agent must be assigned a *credit share* value 1.

4 Experiments and Results

We have implemented agent based wave computation in an event-driven simulation environment. This section presents the results of various experiments conducted to study the control parameters for an efficient wave algorithm. As indicated earlier the agent wave is controlled by *clone* and *merge* operations. Our simulation is simplified by making certain assumptions about the problem domain. We assume that the network is connected without knots, hence the agents may move around in a random motion with no knowledge of the topology and may reach every possible node. We also assume that the agents and the nodes are reliable and that the network is static, i.e. the parameters being probed do not change over time. The computation is assumed to be instantaneous and the network being homogeneous, every agent-node interaction is identical. There is no cost associated with transporting the agents among nodes. A consolidated review of the experiments is presented hereafter.

The first experiment focuses on the beginning phase of the wave computation. A random connected graph of 50 nodes is selected for the experiment. A single agent is placed at a random node. Depending on the connectivity of that node, the agent chooses an edge randomly and moves along that edge to a neighboring node. If the agent gathers new information from the node, it is cloned, i.e. *deterministic cloning*. Each of the resulting agents move to different nodes and the computation continues thereafter. On meeting another agent at a node the two agents merge, i.e. *deterministic merging*. The computation stops when the agent population has converged to a single agent, which contains complete information.

From Figure 2 it is evident that the agent population increases initially due to the *clone* operation and once it reaches a peak, there is a gradual decrease in population due to the *merge* operation. The figure also indicates the time when at least one agent has complete information. In principle, the task of collecting information is completed at this point, however, agent activity continues until convergence of the agent population. This period of time marks unnecessary resource overhead and techniques to reduce this overhead were investigated. The simulation was repeated over different random number sequences and the results obtained showed little variation in the principle behavior. If *deterministic cloning* is employed, it is possible for the agent population to exceed the number of nodes in the graph. This is due to the fact that every agent is free to clone when it arrives at a node that the agent had previously not encountered. However, the steep decline in the curve indicates that such behavior is followed by large scale merging.

The experiment was repeated for different cloning probabilities. These experiments showed similar basic behavior wherein the agent population initially increased and then there was a steady decline in the number of agents. However, for lower cloning probabilities the peak number of agents in the network was smaller than that for higher cloning probabilities. Here the agents do not create clones every time they visit a node which they had not visited earlier. Also, due to lower number of agents in the network, the agents take more time to merge

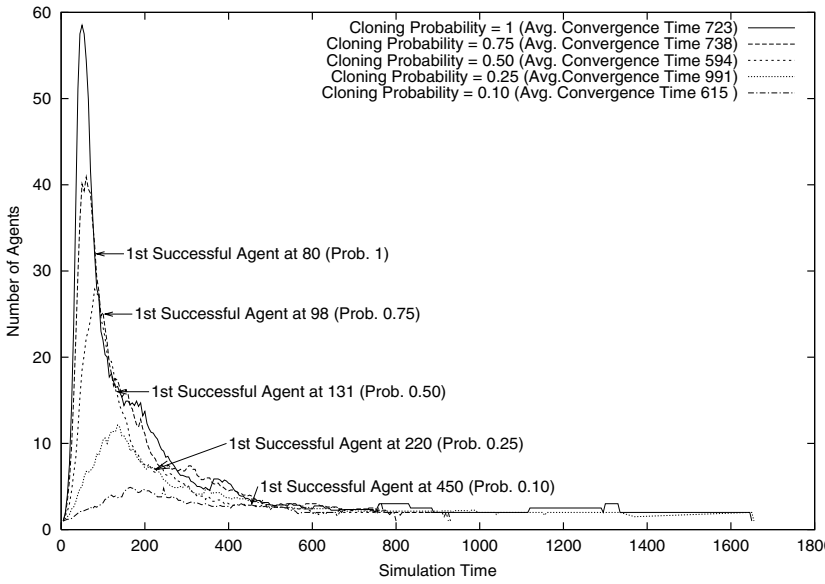


Fig. 2. Probabilistic cloning

with other agents and get complete information about the network. However, there is no apparent correlation between the cloning probabilities and time until the agent population has converged to a single agent.

Additional experiments were conducted in an attempt to reduce the unnecessary resource overhead by inhibiting the cloning ability of the agent. After an agent has visited a number of nodes without acquiring new information, its cloning ability is disabled. While the agent is still free to move in the network, collect information from nodes and merge with other agents, its ability to procreate is eliminated.

The results in Figure 3 indicate that this technique significantly reduces the time for convergence of the agent population to a single agent. This is due to the fact that there are fewer agents in the network to converge, since the agents do not clone after visiting few nodes where they do not acquire new information. Here we are attempting to reduce the number of agents created after an agent has not added information to its knowledge base, which indicates that the agent has visited quite a few nodes in the network.

After cloning is disabled, a situation may occur in which a few agents continue to traverse in the network without merging. To improve upon such behavior, we explored the possibility of forced termination by using the agents knowledge of a path to the origin node of the wave computation. This technique requires that the agents maintain a valid return path to the node at which the computation originated. Once the agent is disabled from cloning, it uses this path information to return to the origin node. On arrival at the origin node, the agent waits to merge with other arriving agents. The computation continues until all agents

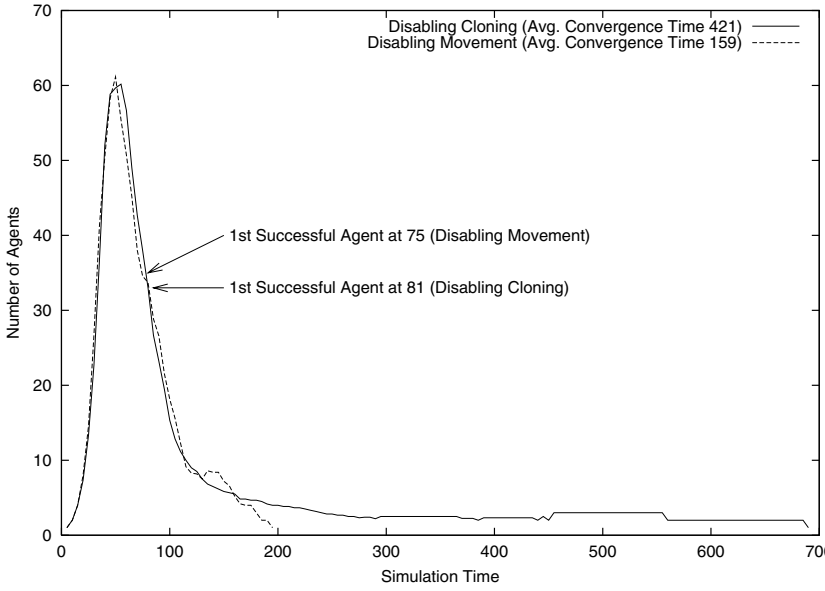


Fig. 3. Disabling cloning and agent movement.

return to the origin node, where they merge with other agents. Figure 3 shows the results when such a technique was implemented. The computation converges significantly faster than the situation where agents were disabled from cloning as indicated in Figure 3.

5 Conclusion and Future Work

In this paper we have presented methods to control the resource overhead for an agent based computation in a fundamental network monitoring application. Since the task of network monitoring is accomplished by employing a population of mobile agents, there is inherent parallelism in such an approach. This is achieved by mapping the agent approach on to the concept of a wave computation. Agents go through a series of clone and merge operations, giving rise to a dynamically varying agent population. The complexity or size of the computation is constantly controlled by a set of agent population control parameters. The computation terminates when the agent population converges to a single agent with complete knowledge obtained from all nodes in the network.

Simulation experiments were based on a set of assumptions such as the network information being static and agent knowledge being reliable. Some of these assumptions may be relaxed to gain further insight into the behavior of multi-agent approaches in network management. For example, this paper assumes a static network topology. In case of a dynamic network, the completion of computation can not be defined. In such cases, the agent population would not decline

to a single agent but would need to oscillate about a suitable average value. The goal is to maintain an appropriate number of agents in a dynamic network at any time, whose population can be increased or decreased according to network dynamics. To accomplish this the agents may have to incorporate time as a parameter for their decisions. There is also a possibility that the agents do not complete the task due to lack of knowledge about the system being investigated. For example, there may be situations in which the agents do not visit all nodes in the network before declaring completion of the computation.Hence, techniques must be developed to maximize the probability of task completion. In addition to the techniques that have been explored in this paper to control the agent population, one may restrict the the number of clonings an individual agent may perform. Also, use of the agents' knowledge about the system in making migration decisions and development of a communication scheme for better orientation of agents need to be considered and investigated.

References

1. Edsger W. Dijkstra, C.S. SCHOLTEN, *Termination Detection for Diffusing Computations*, Information Processing Letters, Volume 11, Number 1,Aug 1980.
2. Nelson Minar - Kwindla Hultman Kramer - Pattie Maes, *Cooperating Mobile Agents for Mapping Networks*, Proceedings of the First Hungarian National Conference on Agent Based Computing.1998.
3. Nelson Minar - Kwindla Hultman Kramer - Pattie Maes, *Cooperating Mobile Agents for Dynamic Network Routing*, Software Agents for Future Communications Systems, Springer-Verlag, 1999, ISBN 3-540-65578-6.
4. Gerard Tel, *Introduction to Distributed Algorithms*, Cambridge University Press,1998,
5. Jeff Matocha, Tracy Camp, *A Taxonomy of Distributed Termination Detection Algorithms*, The Journal of Systems and Software 43 (1998) 207-221.
6. Germáman Goldszmidt, Yechiam Yemini, *Distributed Management by Delegation*, Proceedings of the 15^{th} International Conference on Distributed Computing Systems, June 1995.
7. Friedement Mattern,*Global Quiescence Detection Based on Credit Distribution and Recovery*, Information Processing Letters 30, pp. 195 - 200, 1989
8. Mario Baldi, Silvano Gai, and Gian Pietro Picco. *Exploiting code mobility in decentralized and flexible network management*, Proceedings of the First International Workshop on Mobile Agents, Berlin, April 1997.
9. Anselm Lingnau and Oswald Drobnik. *An Infrastructure for Mobile Agents: Requirements and Architecture*. Proceedings of the 13th DIS Workshop, Orlando, Florida,September 1995.
10. Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko, and Daniela Rus. *Mobile agents in distributed information retrieval*, In Matthias Klusch, editor, Intelligent Information Agents, chapter 12. Springer-Verlag, 1999. To appear as ISBN 3-540-65112-8.

A Design for JTrader, an Internet Trading Service

Marcelo d'Amorim and Carlos Ferraz

Universidade Federal de Pernambuco
Centro de Informática
Caixa Postal 7851, 50640-970, Recife-PE, Brazil

Abstract. Over the last few years, Service Discovery Protocols (SDP) like SLP, Jini, UPnP and Bluetooth demand special interest on scientific community and industry since they promise a new engagement opportunity between services and clients. Despite the fact that these protocols were first concerned with connecting devices, they also play an important role on information services trading which is very relevant considering Internet's large scale. This paper presents the design of JTrader, an Internet Trading Service based on Jini Technology whose main purpose is to address root problems around service trading on the Internet.

1 Introduction

Currently, mobile communication is being considered a very promising market due to the rapid popularization of cellular telephones and devices such as laptops and PDAs. In the near future, it is expected a strong deployment of services in order to fulfill this demand. Location transparency is not a recent problem but has assumed great importance in an environment where clients change their place very often. With service discovery protocols, devices (or simply clients) find network services by its properties and services advertise their availability in a dynamic way. On the other hand, assuming the Web moves from document to program containment, this network is likely to be the greatest distributed system ever seen with numerous programs for several kinds of tasks. This diversity implies a greater number of bindings among programs. To guarantee system's bindings stability, components must be super-reliable, which is very hard to achieve. The alternative is to build distributed applications layered over an intelligent service discovery framework whereby the bindings between components can be (re) discovered at runtime, and do not have to be static – this is called *service trading*.

This work presents a coarse-grained design for an Internet service federation system, which helps users to find and use global services based on its characteristics. As we will see, components attempting to publish services over the Internet should notify their offers to the global trading federation by means of distributed agents, which serve as monitors, reporting to the centralized federation every relevant event such as a network failure or some modification on a remote service's state.

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 159–166, 2001.

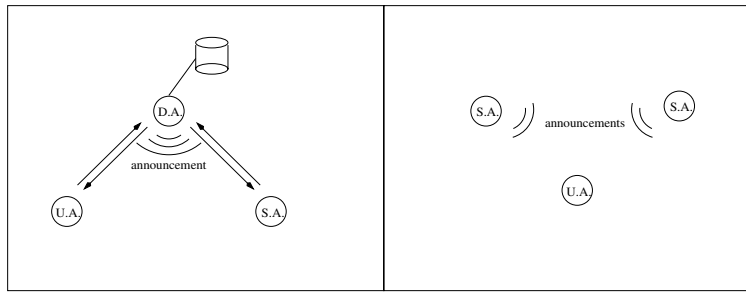


Fig. 1. Trading architectures – (a) DA-based and (b) peer-to-peer approaches

2 General SDP Architectures

This section does not intend to present a tutorial on SDP technology, but defines a structural model on which these protocols are based, thus generalizing the solution as long as it is presented at a high abstraction level.

Analyzing trader solutions under a structural point of view, we identify three basic components: directory agents (DA), user agents (UA) and service agents (SA). The DA component is responsible to store service offers and execute queries by matching the stored offers against query constraints, which can result in a set of service items. A service offer describes an available service implementation and can aggregate some important properties like the service location, for example. In addition, a service offer provides a means to access the service implementation it describes. The offer should store a remote reference to an object, like CORBA [1] does; an IP address and port where the service is running, like SLP (Service Location Protocol) [2] does; or a full-fledged proxy object that directly or indirectly implements the service interface, like Jini [3] does.

UA components are responsible to elaborate queries on behalf of users and submit these queries to the trader which may answer with a non-deterministic[1] set of service items that satisfy query constraints; so, it is up to this component to select the best item(s) returned. On the other hand, while a service implementation keeps running, the SA component is in charge to maintain service offers consistent on DAs. Figure 1.a shows the relationship among these components. In order to perform these tasks, user and service agents need to find a suitable DA to, respectively, search for service instances and keep available service offers. In fact, a *discovery* protocol enable these components to find directory agent instances.

The most common used discovery protocol version makes use of unicast communication; however, it can leads to lack of location transparency as UA and SA components are statically bound to a physical address where the DA instance hosts. As a way to achieve location transparency by not addressing messages to

[1] Depending on the query method used. Some methods are synchronous and may impose timeout boundaries.

some specific machine and port, some SDP protocols implement discovery using multicast communication. Actually, multicast discovery protocols can be active or passive. In the first case, a user agent start the protocol sending a request message to some known multicast endpoint and receive callback messages that running DAs, which listens to that endpoint, send. System administrators very often define groups for DA's participation as a means to segregate service categories. Since components send the groups they are interested in request messages, only those that participate in the informed groups will respond. In fact, unicast discovery is also a particular kind of active discovery. On passive discovery, DAs regularly send announcement messages, also known as "I am alive" messages, representing its availability. In this case, UAs and SAs select among DAs instances announced, those representing some group of interest. Indeed, passive discovery enable components to test DA's availability in a similar mechanism as the negative acknowledgment in request-response protocols [4].

Although some SDP protocols like Jini and CORBA Trader enforce the use of DAs, in some protocols they are optional or even do not exist, for example UPnP (Universal Plug and Play) [5]. So this work envisions two trader configurations as shown on Figure 1: one using DA (1.a) and the other (1.b) without them. We figure out the peer-to-peer configuration as a refinement of the DA-based architecture where the remaining components assume tasks of the absent DA. On peer-to-peer configuration SAs frequently announce their availability by means of multicast announcement messages in a very similar approach as passive discovery. On the other hand, UAs select only those services of interest.

Peer-to-peer is simpler than DA-based trader and, in some contexts, is also considered more reliable since DAs introduce another failure point and also the opportunity to inconsistently represent service properties. Moreover, DA represents a single communication point which parties must contact prior to start direct communication with a target service, thus introducing some overhead. On the other hand, regarding to communication overhead, peer-to-peer traders span multicast messages proportionally to the number of network services. In DA-based traders, this number is proportional to the number of DAs, which is theoretically less than the number of services. In addition, a directory-based architecture allows direct reach-ability of the name services by the unicast discovery protocol, which has an important role on the current Internet where there not enough support to multicast communication. Therefore, peer-to-peer traders are being considered well suited to small-scale networks with a few numbers of services, like home or automobile networks. UPnP and Bluetooth [6] are examples of peer-to-peer traders, the later being specific for short range wireless networks. On the other hand, repository-based (DA) traders seems better suited to large-scaled networks, like the Internet. [7] makes a comparison among Jini and CORBA traders, the two main representatives of repository traders where is posed that Jini presents better tradeoffs than CORBA in regard to Internet trading.

3 JTrader Design

This section presents a design for a trading service federation over the Internet where services are likely to be installed every day. Trading seems to be a very appropriate model to be used in this dynamic environment as it is based on an asynchronous and very decoupled distributed programming model that allows clients to discover services with very few information about them. For instance, there is no implicit order for starting components: first the server, next, the clients, which would mean a difficult item of configuration, considering the Internet global scale. Furthermore, trading over large-scale networks enables users and client programs to use services wherever they are – there's no locality constraint concerning the service.

JTrader presents a Jini-based software solution that enables services to be registered transparently in an Internet service federation, and almost without system administration efforts. The system is in charge to gather public remote service proxies on the enterprise LANs and register them on the federation, providing a means for localizing global services wherever they are hosted. As long as users and programs retrieve service offers on the federation, communication conveys directly among peers. In contrast to strict peer-to-peer architectures, this approach somehow centralizes service offers in order to make possible universal reach-ability. Actually, worldwide famous peer-to-peer applications like Napster and ICQ also perform some kind of centralization [8]. For example, you should register on the Napster server the directory where you allow access to music files, and you also should set up email and nickname when using the later program for the first time. While proxies stored in service offers do not maintain state, centralized data only represent a link to the remote service[2]. From now, the JTrader coarse-grained architecture and core components are presented:

- *JTrader Federation* – JTF. The federation manages sets of DAs, which in Jini are called Lookup Services or simply Lus, to register remote service offers. As a consequence, it defines a policy governing how this registration proceeds as a means to provide load balance and scalability. In fact, this is the core system component on which globally accessible services are registered and also users and programs should contact to search for interested services.
- *JTrader Federation Adapter* – JTFA. As a means to provide seamless federation accessibility, this component isolates from clients the hard task of finding and keeping a valid reference to the remote federation object. As long as JTF instances crash or change their place, this component is in charge to find a new valid reference.
- *JTrader Enterprise Agent* – JTEA. Installed on private local networks, this component is a remote stationary agent, which listens for local services availability in order to publish their offers on JTrader network.

[2] Although service properties are also considered part of the service state, JTrader keeps them consistent with the service remote implementation.

– *JTrader Web System* – JTWS. This independent component provides a front-end to Web users search and access services available on the federation by its user interfaces.

4 Federation Details

Designed for large-scale use, the federation component – JTF – should meet scalability, availability and performance requirements. The system should keep running even in presence of partial failures, searching for alternative running instances of failed objects and dynamically binding them to the federation – *availability*. The federation should be able to grow, adding new services instances, name servers, and hosts, in order to support access and registration high volumes – *scalability*. Moreover, the federation has to achieve an acceptable overall efficiency by means of conscious balanced compromises – *performance*.

SDP protocols very often provide a means to filter, while in discovery phase; name servers that join on some well known group. Considering system administrators define suitable groups representing service categories some enterprise has, it is possible to discover specific DAs, whose registered services have some similar semantic pattern like "devices". In such a case, the result of discovering DAs joining on this group is that proxies enabled to access devices are registered on them, so query scope is confined. However, this feature is very hard to be achieved on a considerably distributed scenario. Consider, for example, "companyA" uses "devices" to name the devices group, while "companyB" uses "Devices". Actually, this is a hard problem to be resolved both for clients and services. In such a case, clients does not know which string is used to name the device group, and services are enforced to agree on a shared ontology in order to classify their objects.

Even if we assume there are not two groups defined on the federation that share the same category, which is by far an unrealistic and very optimistic scenario, the federation is faced with a hard problem about resource allocation. Usually, enterprise networks define a Lus by group or department in order to scope their queries. An organization by department does not make any sense in a federation, thus we conclude that the federation should have at least on running Lus per enterprise group, which is unfeasible regarding the potential number of groups and the intermittent frequency, which they are used on JTF. Therefore, concerning the federation scale, this behavior is very hard to achieve since the number of potential groups defined in remote networks is some orders of magnitude greater than the number of Lookup Services hosted in the federation. Nevertheless, allowing Lookup Services to participate in multiple and semantically distinct enterprise groups deny the principle of segregating service categories in name services. In fact, JTF does not conform to this principle, it controls which Lus a service is to be registered by providing a front-end interface to the register operation, and thereby the registration is performed transparently to services.

On the other hand, as in regular Jini networks, the federation system should define which groups the Lus actually join, and these are not remote enterprise groups, rather they are groups assigned by system administrator and used to enable load balancing and also raise the system availability. Indeed, Lookup Services joining on the same, here called, physical group are configured to keep the same set of services regardless if they join on distinct user defined groups. In other words, JTrader uses two kinds of groups as described bellow:

- *user-defined group.* This group is managed by JTF and is defined to provide group awareness to JTrader's users. In contrast to the Jini approach of attaching groups to Lus, this approach attaches services to groups as a consequence of using a Lookup Service to store offers from multiple user-defined groups. Actually, these groups are transparent to Lookup Services. On the other hand, JTrader users only have access to this kind of groups.
- *administrator defined group.* A physical and administrative group used to multicast discovery Lookup Service instances. Lus instances joining on the same physical group should keep the same set of services. The consistent replication of services is achieved by tunnels, as shown in [9], installed between Lookup Services joining the same physical group.

Physical groups main purpose is to balance the system load and increase service availability, so they are transparent to users. Therefore, any combination of Lookup Services that has at least one component from each physical group represents the federation and the load balancing solution must guarantee the best-effort selection. As represented in Figure 2, this arrangement enables unprecedented interchange among Lookup Services as long as some instance fail or become somehow inefficient. A directed tunnel between a pair of Lookup Services, causes any modification in one Lus, like a new service being registered, be also replicated to the other. The only constraint is to avoid loop on the tunnel network so that messages would be never dropped. However, this problem can be resolved tagging the messages posted by the initial Lus with an identifier, if the message reaches the source, it will be dropped. Starting new DA instances, attaching these instances to groups, defining tunnels among them, and monitoring services registered are features of an independent administration tool called: JTrader Admin[3].

In Figure 2 three configuration scenarios are presented where five service offers are registered in three distinct user-defined groups. The initial configuration (Scenario 1) has five Lookup Services joining in three physical groups, and also proxy tunnels installed as mentioned earlier: Lus1 $\leftrightarrow$ Lus2, Lus3 $\leftrightarrow$ Lus4. The second scenario presents an automatic reconfiguration after a Lus failure whereby the replicated Lus start to respond for services registered in the failed one. The third scenario also presents a Lus failure situation, but in this case the Lookup Service had no replication.

In a Jini private network, a service can register to receive events on Lus failure and try, in response, to discover a new available one that joins the same

[3] At this moment, this auxiliary tool is not implemented.

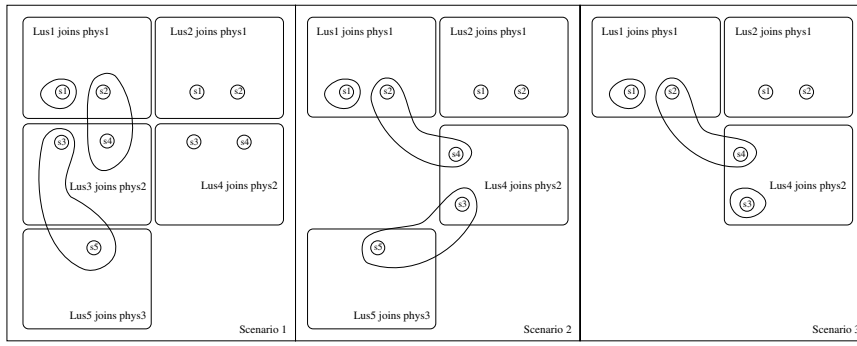


Fig. 2. User-defined and physical groups

group. In contrast, JTrader uses a different approach in order to provide service availability and reduce communication with the federation. Using tunnels, JTF makes a best effort to keep services proxies available. If a proxy is unregistered it is due to failing in renewing a lease[4] or no more Lus available on the physical group, as presented earlier.

As mentioned earlier, the JTF component can be considered as a special case of discovery protocol as it provides a means to access DAs. The federation interface also defines a `register` operation which could seem unnecessary considering the Lookup Service interface – `ServiceRegistrar` – actually defines a similar operation. However, using the Lookup Service `register` method makes sense in environments which users have some influence over DA creation like enterprise networks; but this is not the case on wider networks, which user-defined groups are created on a demand basis. When searching a Lookup Services of a given user defined group for the first time, probably, no instance will be returned by the federation, thereby service offers could not be registered if the `register` operation were not provided on JTF interface. In addition, this operation enables JTF to perform bookkeeping tasks with regard to the user-defined groups it manages. The JTF interface is defined as follows:

```
public interface IJTrader {
   public ServiceRegistration register(String group,
                                       ServiceItem item,
                                       long leaseDuration)
       throws RemoteException;

   public ServiceRegistrar[] discover(String group)
       throws RemoteException;
}
```

Once services are successfully registered on JTF, Lookup Service proxies may be retrieved by calling the `discover` method. Therefore, as Lookup Services are

[4] Lease renewal is handled by the adapter component detailed in the next section.

localized, client components do not need to interact directly with the federation anymore, thus alleviating JTF from concurrent access. Notice that more than one Lus proxy may be returned in a `discover` call since Lookup Services may become overloaded and the federation applies an heuristics to balance the load, as discussed on the next section. In fact, this decision may affect Jini client component as the Lus instances returned represent disjoint sets of services, thereby a client should query each Lus until find the intended service. However, we will see the system can be configured to not use this feature.

5 Concluding Remarks and Future Works

In order to enable Internet service dynamic discovery and binding, this work introduced current SDP technologies, presented some scenarios of use, and finally presented a coarse-grained design for JTrader, an approach to service trading over the Internet. The JTrader system is made of four components: the *JTrader Federation* core component manages a federation where all currently available services are registered, providing a high scalability and reliable service; the *JTrader Federation Adapter* provides a means to programs transparently access the federation on remote locations; the *JTrader Web System* is a Web front-end that provides tools to search for services on that federation and using them; and finally *JTrader Enterprise Agent* provides a means to enterprises publish their services almost without administration.

Concerning this work's main purpose is to define a coarse-grained Internet trading architecture, network issues, like connections behind firewall, are considered in [10]. Performance analysis is to be done as a future work.

References

1. Object Management Group: CORBA OMA Specification. June (1986)
2. Kempf, J., Pierre, P.: Service Location Protocols for Enterprise Networks. John Wiley & Sons, (1999)
3. Arnold, K., O'Sullivan, B., Scheifler, R., Waldo, J., Wollrath, A.: The Jini Specification. Addison-Wesley, December (1999)
4. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems : Concepts and Design. 3rd. Edition, Addison-Wesley, August (2000)
5. Microsoft Corp.: Universal Plug and Play Device Architecture Reference Specification. Available at http://www.microsoft.com/hwdev/UPnP, November (1999)
6. Haartsen, J. et al.: Bluetooth: Vision, Goals, and Architecture. Mobile Computing and Communications Review, October (1998) 38-45
7. d'Amorim, M., Ferraz, C.: A Comparative Analysis of Trading Approaches. Submitted to the 19th Brazilian Symposium on Computer Networks – SBRC2001. Florianpolis, Santa Catarina, Brazil. May 21-25 (2001)
8. Shirky, C.: What is P2P ... And What is Not. Available at http://www.openp2p.com/lpt/a/472, The O'Reilly Network, October (2000)
9. Edwards, K.: Core Jini Sun Microsystems Press, June (1999)
10. d'Amorim, M., Ferraz, C.: Leveraging Security in Internet Distributed Applications Submitted to SBRC2001

Computer-Supported Deliberations for Distributed Teams

Jacques Lonchamp and Fabrice Muller

LORIA, BP 254, 54500 Vandoeuvre-lès-Nancy, France
{jloncham,fmuller}@loria.fr

Abstract. In our terminology, a "deliberation" is a distributed collaborative process, more or less spontaneous, structured, and complex. This process can include both individual and collective activities, synchronous and asynchronous phases. But in every case, its core part is the negotiation of some result through argumentation. This paper describes a comprehensive flexible generic support for distributed deliberations: motivations, design choices, current Java prototype, and usage through a collaborative document inspection example.

1 Introduction

In our terminology, a "deliberation" is a distributed collaborative process, more or less spontaneous, structured, and complex. It can include both individual and collective activities, synchronous and asynchronous phases. But in every case, *at the heart of a deliberation is the negotiation of some result by exchanging arguments*. The collective elaboration of some artifact (e.g. a requirements document) and the collective assessment of an individually proposed artifact (e.g. a code inspection) are classical examples of deliberation types. In classical organizations, such processes heavily rely on face to face meetings. In new distributed and temporary forms of organizations (virtual teams, virtual enterprises, multinational research projects), there is a strong need for a specialized computer support for such processes. We claim that no comprehensive and flexible computer support is currently available. Most current tools emphasize a single aspect of cooperative work: communication (e-mail, chat, bulletin boards, Internet telephony, video conference, …), coordination (workflow management systems), or collaborative production (shared editors, shared document repositories, …). Deliberations require a comprehensive support for all these aspects, and it is not easy to build a satisfying support with fragmented tools. Integrated environments could be better, but most of them are either restricted in their scope (e.g. synchronous environments based on the room metaphor, workflow environments with a loose integration of groupware), or so complex, that they appear not suitable for temporary and rather light weight processes. Our research project aims at providing *a comprehensive and flexible generic support for distributed deliberations*.

The rest of the paper is organized as follows. Section 2 introduces the requirements. Section 3 summarizes the main design choices. Section 4 describes DOTS ('Decision

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 167–174, 2001.

Oriented Task Support'), our current Java prototype, its architecture, and shows its usage through a collaborative document inspection example.

2 The Requirements

There exits a wide range of deliberation types. For instance, a brainstorming process can include a single synchronous session, where participants propose and discuss ideas. Another brainstorming process, can include individual parallel activities for ideas proposal (either private, public, or semi public), followed by a synchronous phase for collective discussion of the individual proposals. Our aim is to *build a generic system which can support this wide range of processes*. Deliberation models should be parameters of the generic system (R1). The system should support structured processes, with both individual and collective activities, and their coordination (R2). The system should support both synchronous and asynchronous phases (R3). In many cases, *the process organization is defined opportunistically by the participants when they participate it*. For instance, in a multi-disciplinary design solution assessment in the aeronautical field [8], the participants can choose first an analytical (criteria based) assessment of the initial solution. Then, if no convergence occurs, and if other solutions are proposed, they can continue with comparative assessments of the alternative solutions. They can also choose analogical assessments with past solutions. The process model should define different strategies for reaching the final goal, letting the participants (all of them or only a 'process manager') make the tactical choice of the effective process at execution time (R4).

At the heart of a deliberation is the negotiation of some result by exchanging arguments. The system should support the core argumentation and decision activity (R5). Argumentation can take different forms: in some design activities, arguments are related to precise criteria and can be weighted quite precisely. In other applications arguments are less formal and no weighting is possible. The system should provide a flexible argumentation and decision scheme (R6). In many deliberations, the importance of an argument is related to the person that emitted it and to the role of this person in the organization ('arguments of authority'). The system should support actor and role identification (R7).

The system should support distributed work, because temporary alliances of partners are the typical target of the system (R8). Due to the temporary nature of these networks, the system should be easy to install and use (R9). In distributed teams, interaction can occur either indirectly, by sharing documents, or directly, through dedicated synchronous or asynchronous communication channels. Direct communication can have different levels of formality. The system should support all kinds of communication: direct/indirect, synchronous/asynchronous, formal/informal (R10). It should manage a range of possible artifacts (R11). For instance, problems, solutions, constraints, in a solution assessment deliberation process. In distributed teams, participants need both guidance and awareness facilities, for answering the classical who?

when? where? what? how? questions. The system should provide those facilities, together with less usual argumentation and decision making assistance (R12).

3 The Main Design Choices

3.1 A Meta Model for Describing Deliberation Models

Our deliberation support system is a generic process-centred system (R1). For building a dedicated environment, a deliberation model describing a set of types and their relations is written with a specific modelling language. This model is instantiated, partly with an instantiation tool, for instances that are necessary for starting the task, and partly during the model execution. Our modelling language is defined by a meta model, extending a classical workflow process meta model with a decision-oriented view [4]: we call our approach *"fine grain decision-oriented task modelling"*. The meta model includes:

– *a process view*, with task and phase types; the instantiation builds a network of phase instances, with precedence relationships;
– *a product view* (R11), with artefact types, specializing text, list, graph, or image generic types, and fine grained component types, specializing list element, graph node, or graph vertex generic types. Application tool types mirror the specialization of the artefacts types, with specializations of text viewer, list viewer, or graph viewer; each phase type grants access to application tool types;
– *an organizational view* (R7), with role and actor types;
– *a decision-oriented view* (R5), describing collaborative work at a fine grain level; each phase type is defined by a set of issue types, which must be solved either individually or collectively. Each issue type is characterized by a set of option types, which describe the different possible resolutions. At run time, one option is chosen trough an argumentation process. Each option type can trigger, when it is chosen, some operation type, which can change some artefact, or change the process state (e.g. termination of a phase), or change the process definition itself.

3.2 Dynamic Task Model Refinement

A model designer can have to describe several strategies for performing the task. In our system, each strategy is described within a dedicated phase type. The choice between them takes place at run time, within a model refinement phase. The model refinement phase type includes a model refinement issue type*, with one option type for each strategy*. At run time, one of these option is chosen (either individually by someone playing a 'process manager' role, or collectively). The corresponding strategy is deployed, by instantiating new phase instances and new phase precedence (R4). By this way, artefact and process elaboration are similar, and the system can provide the same kind of assistance and guidance for both aspects.

3.3 Argumentation and Decision Support

Participants can express arguments about the different options of an issue and qualitative preferences between arguments. At every stage of the argumentation, the system computes a preferred option, *through an argumentative reasoning technique* similar to those described in [2, 3]. The issue, its options, the arguments 'for' and 'against' the options, the arguments 'for' and 'against' the arguments form an argumentation tree. A score and a status (active, inactive), which derive from the score, characterize each node of the tree. The score of a father node is the sum of the weights of its active child nodes which are 'for' the father minus the sum of the weights of its active child nodes which are 'against' the father. If the score is positive, the node becomes active, otherwise, it becomes inactive. Only status propagates in the tree because scores have no global meaning. Without preferences constraints all nodes have the same weight. Leaves are always active. The preferred option computed by the system has the maximum score. Preference constraints are qualitative preferences between two arguments: the importance of one argument is compared with the importance of the other. The aim of constraints is to define different weights for arguments in a very flexible and revisable way (R6). To each constraint is attached a "constraint issue" with three positions: 'More important than', 'Less important than', 'Equally important than'. Thus, all constraints are subject to debate and can change their meaning dynamically. A constraint is active if both its source and its destination arguments are active, if one of its option is chosen (with a score strictly higher than the others), and if it is consistent with the other constraints. Constraint consistency is evaluated (with a path consistency algorithm) each time a constraint is created or becomes inactive. Weights are computed by a heuristics which gives a medium value for arguments not participating to constraints, a higher value for arguments which dominate other arguments and a lower value for arguments which are dominated by others.

Argumentation is useful both for individual and for collective issues. For individual issues arguments explain some rationale (e.g. in the demonstrative example, an individually proposed solution should come with a detailed rationale; on the opposite, a new constraint could come with a single argument as its justification). For collective issues argumentation prepares the resolution. In most cases the decision is kept separated from the argumentation result. Each issue type has a resolution mode property, which defines how issue instances are solved. For instance, 'autocratic without justification', when an actor playing a given role can take his own decision, 'autocratic with justification', when the decision maker has to argue in order to make his/her choice equal to the preferred choice of the system, 'democratic', when the preferred option computed by the system is chosen.

3.4 Synchronous and Asynchronous Work Support

In a typical deliberation, some result is negotiated collectively. At the beginning of the process, participants can express their opinion asynchronously. Synchronous sessions are useful when the argumentation becomes complex and when no rapid convergence occurs. Our deliberation system supports both working modes and provides adapted

guidance and awareness. The model designer can customize the way synchronous sessions are organized. First, synchronous sessions can be managed as artefacts, i.e. created through an issue resolution, by a given role, and accessed by the other participants through application tools (meeting list viewer, meeting reminder). Secondly, the organization process can be supported through issue types such as 'Call for organization', 'Answer organization', and 'Call for participation': the operation types that are triggered by these issue resolutions can use direct communication channels (R10) (e.g. dynamically built e-mail messages, internally managed by the system). This organization process can be described in a 'Meeting organization' phase type whose instances can run in parallel with other asynchronous phases. The system also provides *awareness indicators (R12) for deciding which issues require a synchronous discussion*. These indicators are computed through an analysis of the valuated argumentation tree and are directed towards participation (who has already participated? when ?), divergence of opinions (who supports which option? by correlating participant interactions and option score evolutions), stability (is the preferred option computed by the system stable or not?). During asynchronous sessions, another awareness facility answers the 'what's new?' question, highlighting entities that have changed since the end of the previous connexion of the same user. During synchronous sessions, several synchronous communication channels (chat, whiteboard, vote widget) are provided (R10). They can be used for instance for 'meta discussion' about the session (e.g. decision to skip to the next issue).

4 DOTS Prototype

4.1 DOTS Architecture

DOTS ('Decision Oriented Task Support') is our current prototype. It is completely developed in java for portability reasons (R8) and runs on the Internet (R9). It has an hybrid architecture with a traditional client/server architecture for asynchronous work and direct inter-client communication for synchronous work. The server is built on top of an open source multi user object base (www.sourceforge.net/projects/stored objects), which stores each model and all the corresponding instances (R11). The synchronous infrastructure (sessions, channels, clients, tokens) is provided by JSDT (www.java.sun.com). During synchronous sessions, one user (with the pre-defined 'Moderator' role) extracts already asynchronously started argumentations from the database and stores their results after the synchronous discussions.

4.2 DOTS Development Environment

DOTS comes with a complete development environment. All tools are developed in java. First, a deliberation task model is written with DOTS modelling language and DOTS Integrated Development Environment (IDE). This model is compiled into java classes by the IDE. Together with the java classes of the generic kernel, they consti-

tute a dedicated environment. All these classes are compiled with a java compiler and a persistence post processor to become a persistent executable environment. Thanks to DOTS Instantiation tool, initial objects are created in the object base (actors, deliberation instances, initial documents, etc). The instantiated model can also be checked for completion by DOTS static analyser. Execution can start through the asynchronous client, which is independent of any deliberation model. The synchronous client is called from the asynchronous client. End users only manipulate the clients and the instantiation tool. Development of new deliberation models is performed by specialists (tool providers), who master the task modelling language.

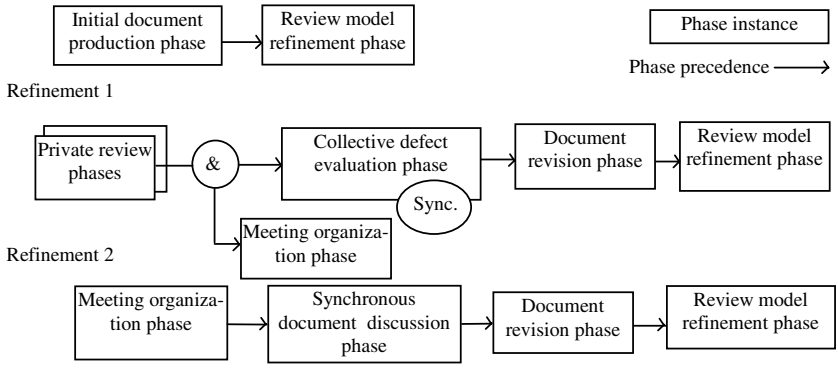


Fig. 1. The process view

4.3. DOTS Usage

The Deliberation Model – The Process View

We explain DOTS usage through a collaborative document inspection model. Fig.1 shows the instantiated model. The 'Initial document production' is an individual phase instance, where an initial version of the document is produced. In the 'Review model refinement phase', the 'process manager' (or all the participants) choose dynamically how to perform the first review. In Fig.1, the first refinement defines a structured review beginning with parallel private reviews. The 'Collective defect evaluation phase' starts in asynchronous mode, and terminates with a synchronous session (virtual meeting) for discussing the more controversial defects. Its organization is defined in a dedicated phase running in parallel (AND operator). The 'Document revision phase' is individually performed by the author and produces a new version of the document. The second refinement defines a simpler process starting with the organization of a synchronous public discussion of the possible remaining defects. The third refinement terminates the deliberation.

The Deliberation Model – The Decision View

Each phase type is basically defined by the issue types it contains. Table 1 summarizes the main issue types of the model. The most important one is the 'Challenge defect'

issue type, where individually proposed defects are collectively discussed by all the inspectors. Individual issue types can have a single option type or multiple option types. In the latter case, the issue resolution maps an individual choice, the arguments being the rationale of the choice. In the former case, the resolution can either be completely automatic, as soon as the issue is raised (this case corresponds to an individual action), or can require at least one argument, as an action rationale. Collective issue types have at least two option types for discussion. The way to solve the issue is defined through the resolution mode defined in the task model. For instance, the 'Challenge defect' issue type can have an autocratic resolution mode (for the 'process manager' role), and the 'Terminate Phase' issue type of the 'Collective defect evaluation' phase can have a democratic mode.

Artifact types managed by this process include at least 'Document, 'Defect', and 'Correction'. Several application tool types are available, such as 'Document viewer', 'Proposed defect viewer', 'Accepted defect viewer', 'Rejected defect viewer', 'Correction viewer', and perhaps a 'Check list viewer'. All of them are specializations of 'Textual viewer' and 'List viewer' generic types. Finally, role types include 'Process manager' (the deliberation manager), 'Inspector', and 'Author'.

Table 1. Main issue types

Phase types	Issue types	Characteristics	Option types	Comments
Review model refinement	Choose refinement	Individual (process manager) or collective	Choose model 1, Choose model 2, …, Terminate	Dynamic model refinement
All phase types	Terminate phase	Collective or individual	Continue, Terminate	Phase termination discussion
Private review and Collective defect evaluation	Propose defect	Individual	-	Defect proposal and description
Collective defect evaluation	Correlate defects	Collective (inspectors)	Correlate, Dissociate	Defect similarity discussion
Collective defect evaluation and synchronous discussion	Challenge defect	Collective (inspectors)	Keep, Reject	Defect discussion
	Suggest correction	Individual	-	Defect correction proposal
Meeting organization	Call for organization	Individual (process manager)	-	Propose some possible dates
	Answer for organization	Individual (inspectors)	-	Give preferences
	Call for participation	Individual (process manager)	-	Define the final date

5 Conclusion

In new forms of distributed organisations, many issues are discussed through deliberation processes. These processes can vary in complexity and formality. This paper describes a comprehensive and generic support for a wide range of deliberations types

within distributed teams, which integrates concepts and techniques coming from workflow management systems, synchronous and asynchronous groupware, argumentation and decision making systems.

DOTS may supersede many previous systems, such as argumentation tools [2, 3], inspection tools [6], generic inspection environments, such as CSRS [9] and ASSIST [7]. CHIPS [1] is the closest prototype from DOTS that we know, mixing process support and collaboration support. CHIPS has a broader scope (flexible business processes), but provides no specific support for argumentation and decision.

Several aspects of DOTS need further work. First, the integration with distributed document management system on the Internet should be considered. Currently, DOTS provides no support for managing the shared artifacts. Secondly, if reuse of artifacts and deliberations is theoretically possible by keeping all terminated deliberations in the object base, we feel that a better alternative could be to feed the enterprise memory with these deliberations, in a more human readable form. A bi-directional interface with enterprise memory systems (for instance based on XML technologies) could be a very promising extension to the current system. Finally, we must assess the system with real users. A previous version has already be evaluated by students playing code inspections [5]. Now we are working with specialists of cognitive ergonomics for modeling evaluation of solutions during co-design processes [8]. We plan to produce a dedicated environment by parameterizing our generic kernel with such a model.

References

1. Haake, J..M., Wang, W.: Flexible support for business processes: extending cooperative hypermedia with process support. Information and Software Technology, 41, 6 (1999).
2. Karacapilidis, D., Papadias, D.: A group decision and negotiation support system for argumentation based reasoning. In: Learning and reasoning with complex representations, LNAI 1266, Springer-Verlag, Berlin Heidelberg New York (1997).
3. Karacapilidis, D., Papadias, D., Gordon, T.: An argumentation based framework for defeasible and qualitative reasoning. In: Advances in artificial intelligence, LNCS 1159, Springer-Verlag, Berlin Heidelberg New York (1996).
4. Lonchamp, J.: A generic computer support for concurrent design. In: Advances in concurrent engineering, CE2000, Lyon, Technomic Pub. Co, Lancaster, USA (2000).
5. Lonchamp, J., Denis, B. : Fine grained process modelling for collaborative work support, Journal of Decision Support Systems, 7, Hermes, Paris (1998).
6. Macdonald, F., Miller, J., Brooks, A., Roper, M., Wood, M.: A review of tool support for software inspection. Proc. 7th Int. Workshop on CASE (1995).
7. Macdonald, F., Miller, J.: Automatic generic support for software inspection. Proc. 10th Int. Quality Week, San Francisco (1997).
8. Martin, G., Détienne, F., Lavigne, E.: Negotiation in collaborative assessment of design solutions: an empirical study on a Concurrent Engineering process. In: Advances in Concurrent Engineering, CE2000, Lyon, Technomic Pub. Co, Lancaster, USA (2000).
9. Tjahjono, D. : Building software review systems using CSRS. Tech. Report ICS-TR-95-06, University of Hawaii, Honolulu (1995).

Hardware Security Concept for Spontaneous Network Integration of Mobile Devices

Igor Sedov[1], Marc Haase[2], Clemens Cap[1], and Dirk Timmermann[2]

[1] University of Rostock, Department of Computer Science
Chair for Information- and Communication Services[†,‡]
{igor,cap}@informatik.uni-rostock.de
[2] University of Rostock, Department of Electrical Engineering
and Information Technology
Institute of Applied Microelectronics and Computer Science[‡]
{marc.haase,dtim}@e-technik.uni-rostock.de

Abstract. In this article we introduce an architecture of a mobile device that enables safe and authenticated data-transmission in a spontaneously configured network environment. The usage of this device is illustrated by a number of examples. The hardware and software components are presented. Particular, we compare Bluetooth and Infrared (IrDA) wireless networking technology, explain the usage of biometrics recognition methods, clarify the choice of the cryptographic module and consider possible platforms for the integration of this trustworthy device into a ubiquitous environment. Subsequently a first realization of the concept will be explained. Referring to feasible possibilities of realization, different attack scenarios together with appropriate solutions are considered.

1 Introduction

Impetuous miniaturization of electronic components leads to rapid growth of the amount of portable mobile devices. Introduction of new technologies increases possibilities of usage Business-to-Business (B2B), Business-to-Customer (B2C) and M-Commerce applications. Secure data transfer between a sender and a receiver is required to preserve privacy of the transmitted information. At the same time, the usage of only internal security solutions in wireless protocols is not sufficient. Mobile units employ mostly radio waves, which can be easily detected, received and decoded in comparison with permanent wire connection. On the other hand, the weakness of existing wireless communication protocols [5,1] is that they make eavesdropping possible and unauthorized usage of transmitted data. Therefore, it is necessary to apply additional trustworthy protocols and algorithms for protection of transmitted data not only on the hardware layer but also on the application layer. Moreover, mobile devices can be easily lost

[†] Supported by Heinz-Nixdorf Stiftung
[‡] Research supported by the SPP "Sicherheit" of the German national research agency DFG

T. Böhme and H. Unger (Eds.): IICS 2001, LNCS 2060, pp. 175–182, 2001.

or stolen, while the true owners of the Digital Assistants (DAs) do not notice it. In this case to exclude potential vulnerabilities, the authentication of the user to his portable device through a Personal Identification Number (PIN) or a biometric recognition procedure should be guaranteed. The main disadvantage of the PIN approach is the usage of mostly four digit numbers. Third parties access to sensible data by searching through all possible PINs. Thus, the PINs length should be now at least 64 bits long. For ordinary people it becomes more and more difficult to remember all their PINs [6]. Our concept provides an identification of a user through a biometric technique, in particular, through a finger print recognition procedure. Biometric tokens are stored within a mobile device in an inaccessible secure memory and never left the device. Moreover, the human biometrics characteristics are not transmittable or forgettable.

In Section 2 we present some reference scenarios which illustrated the use of spontaneously networked mobile devices. Section 3 describes a secure hardware conception, gives an overview of state-of-the-art mobile devices, biometrics recognition methods and cryptographic modules. In Section 3.3 we introduce major properties of modern wireless network technologies and compare Bluetooth and IrDA wireless technologies. Security manager architecture is for management of safety related aspects responsible and presents in Section 3.6. The article closes with a conclusion and directions for future work.

2 Reference Scenarios

The proposed architecture corresponds with the paradigm postulated in [8], that a combination of software, dedicated hardware and reconfigurable logic is the basis for an application-specific optimum. We consider the following reference scenarios for the usage of mobile devices in ubiquitous network market.

- A doctor in a hospital prescribes several medicaments to a patient. However, he does not know which medicaments are available now in the pharmacy. Moreover, the doctor does not know which preparations have already been prescribed to the patient and on which of them he reacts allergically. With the help of digital assistant the doctor can interrogate the required information wirelessly from a server. This scenario requires that doctors, nurses and students are to be distinguished and must to have different access rights. For example, male nurses are allowed to know, when the next medicament must be taken by the patient but not allowed get the other information from disease-history of the patient.
- Suppose that all employees in a hospital have personal digital assistants. If somebody enters or leaves the building, the server registers the time and the date. Besides that, the authorized employees having appropriate access rights to the server, can find out the current location of their colleagues and send to them short messages. Everything is carried out wireless from DAs.

3 Concept

In this section we introduce a concept for a mobile device with the ability for secure spontaneous networking. At the beginning we take a look at state-of-the-art mobile devices and the requirements for trustworthy user devices. Then we explain our architecture for a spontaneous networking mobile device and introduce additional hardware components for the device. At the end of this section we will present a first realization step of a mobile device containing the new functionalities.

3.1 State-of-the-Art Mobile Devices

At the moment several mobile devices are available on the market. There are various groups, i.e., Notebooks, Pocket PC, Personal Digital Assistants (PDA), mobile phones and others. The differences between these devices are CPU performance, system resources, power consumption, peripheral extension possibilities, weight and size. To use these devices as a personal trustworthy user device for security related applications several requirements are recommended.

Regarding to [7] a trusted device should provide an own user interface, i.e., key-pad and display. This ensures that for signing contracts the user can read the contract on his own device and signing the contract with his own keypad. Common Attacks against user terminals will be useless, because manipulating the user interface is not possible. Pocket PC, PDA or mobile phone are devices which provide such an user interface and fulfill the mechanical requirements on a mobile device, that carried permanently. For secure transmissions it is required to combine mobile devices with additional security functions like ciphering, key management, biometric recognition methods. A disadvantage of current available mobile devices is the lack of a wireless communication capability supporting spontaneous networking. We think that IrDA available on most devices is not suitable for this.

Based on the requirements for trustworthy mobile devices with the ability for spontaneous networking we present a new enhanced security architecture for a mobile device that will be implemented.

3.2 Enhanced Security Architecture for Mobile Devices

Our concept provides a mobile device named *SmartBadge*. It consists of a wireless network interface for spontaneous networking in a ubiquitous environment, a cryptographic module for secure data transmission, a processor module for device control and software applications, a secure memory area for private keys and a biometric sensor for user authorization. We will merge these modules and an additional autonomous power supply module to an active device that can be used in a ubiquitous environment. The dimension of this device will be fit to the PCMCIA form factor. It will be controlled by a software architecture especially designed for security.

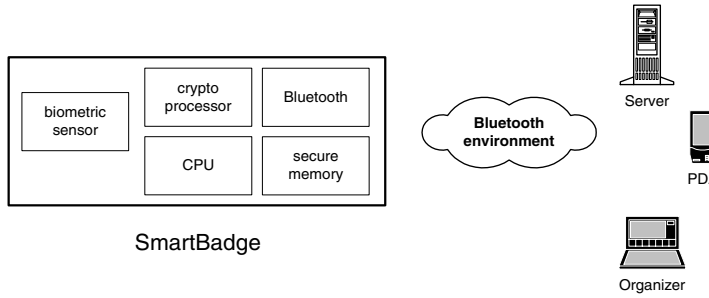


Fig. 1. SmartBadge concept

In difference to [7] we will not implement a user interface in the device because this increases the size of the device and the electric power consumption. For configuration purposes a secure connection to a trustworthy terminal is necessary. This connection can be established over the wireless network interface with strong encryption. Nevertheless we provide an optional hardware connection for a trusted user interface on the card.

Figure 1 shows the architecture of the SmartBadge in a wireless environment especially a Bluetooth network.

3.3 Wireless Network Interfaces

The communication and the data interchange for mobile devices is ideally achieved by wireless network technologies. This ensures the greatest possible mobility. Table 1 shows common wireless network technologies and their major properties [2].

Table 1. Overview of wireless network technologies

Name	Frequency	Range	Data rate
Ricconet	902-928MHz	800m	55Kb/s
IEEE 802.11	2.4GHz	30-360m	10Mb/s(100Mb/s)
DECT	1.8-1.9GHz	100m	522Kb/s(bis2Mb/s)
HomeRF	2.4GHz	50m	2Mb/s
Bluetooth	2.4GHz	10m(100m)	720Kb/s(2Mb/s)
IrDA	light	1-2m	9.6Kb/s-4Mb/s

The use of this technologies depends on the appropriate area network. Referring to the scenarios in section 2 we will concentrate on personal area networks (PAN). IrDA (Infrared Data Association) and Bluetooth are suitable technologies in this case.

IrDA is a popular technology available in most mobile devices. A disadvantage is that IrDA supports connections between two devices with a narrow focused angle of cone maximum 30° [11]. The transfer of the data over infrared-light waves, in spite of the limited range of 1–2 meters between the senders and recipients, enables a "eavesdrop" of reflected infrared-light waves. There is no protection on the link layer. For avoiding any unauthorized use, additional cryptographic algorithms must be implemented in the application layer [4].

Bluetooth is a network technology for wireless data and voice transmissions over short distances up to 100m. An advantage are "Point-to-point" and "point-to-multi-point" connections of Bluetooth devices with a transfer rate up to 721 Kb/s [3]. Up to seven active slave-devices and one master-device form a "pico-network". Each user device from one "pico-network" can participate at the same time in other "pico-networks".

In comparison to IrDA Bluetooth includes different security mechanisms:

- a "frequencies hopping spread spectrum"- technique with a "time division duplex" scheme (FH/TDD). The signal frequency changes up to 1600 times per second in a pseudo-random sequence, which is given by the master,
- error correction methods on link layer,
- four LFSR (linear feedback shift registers) for encoding link layer data,
- a enhanced version of an existing block cipher SAFER-SK128 algorithm for the authentication [3] [10].

In Bluetooth systems there are 79 channels for usage of a complete frequency range. The narrow bands signal carrier hops between the available channels in case of the use of the "Frequency Hopping Spread Spectrum". The narrow signal carrier can be detected with standard receiver, which knows the transferred frequency. The receiver does not have the possibility to grasp the complete data spread, because it receipts only some of FH signals and the consequence of frequency changes is not known.

The sensitivity of the attacker devices and the signal/noise ratio must not be worse than the parameters of the transferring system. Besides the opponent should observe the same sequence of hops, as well as the initial system.

To release a radio jamming, the opponent should block all frequent ranges in which the signal is transferred. In the case of the Bluetooth technology a radio jamming must cover the frequency range from 2.400 GHz to 2.4835 GHz. This requires an increased capacity of the attacker system and therefore can be easily localized.

According to the Bluetooth security architecture and the future plans of integrating Bluetooth in the next generation of mobile devices we decide to use this technology for the SmartBadge concept. The data transmission range of Bluetooth is sufficient for private area networks. The relatively high data transmission rate with small energy consumption and the possibility of "point-to-multi-point" connections enables the usage of Bluetooth technology within the area of wireless communication services.

Unfortunately, actual analyses show security weakness in Bluetooth architecture [5]. To avoid eavesdropping of transmitted data we implement a additional cryptographic module.

3.4 Cryptographic Module

Our concept provides the encoding and decoding of personal data over an independent ciphering module. It supports asymmetrical and symmetrical algorithms, e.g. RSA, DES, TDES. This additional hardware based ciphering module enhances the security functions already implemented in the Bluetooth architecture. The module can be configured and controlled by the software application to encrypt only sensible data.

There are already developed generic VHDL models [9] for the ciphering algorithms. The models can be parameterized according to the data and key lengths defined by the security requirements. We will implemented these algorithms into a field programmable gate array (FPGA). Delegating the encoding algorithm to the hardware components reduces the load on the processor of the mobile device.

3.5 Biometric Sensor and Secure Memory

Today there are too many smart-cards or mobile devices with various PINs and many people are not able to remember all codes. The biometric system applies unambiguous tokens of a person, that depends on physiological or genetic attributes and cannot get lost, nor can be transmitted to other persons. Therefore, the integration of a biometric sensor and a secure memory on the SmartBadge is a most suitable solution for security approach. We recommend for usage a biometric sensor, which unlocks the cryptographic system. After successful authorization of the user the crypto processor generates a session key for further secure wireless communication. Without biometric authorization the private key can not be read from the secure memory and therefore, the session key can not be decoded and the device can not be slated.

Most suited biometric procedure for mobile devices is the fingerprint authentication of the user. The sensor necessary for this integrates easily in the system because of its small size. However in addition to the sensor, the software for the control of the sensor must also be integrated into the system. The processor of the mobile device fulfills this task or the recognition software runs on a processor specially added for this task. Through this the hardware effort is increased on the one hand, on the other hand the biometric recognition method of the system is decoupled completely, and thus possible abuse of the biometric data is eliminated.

3.6 The Function of the Security Manager

Security Manager is the main component for security related aspects. It is responsible for communication with secure hardware and software components and fulfills the function of:

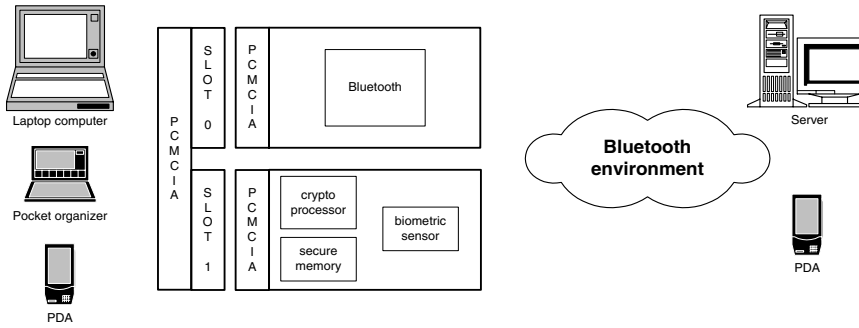


Fig. 2. Concept realization

1. Key management
2. Definition of security levels for all devices and services
3. Distinguish device trust levels
4. Storage identifications information about devices and services
5. Set up Encryption with a necessary key length

The Security manager must define different admission levels for every communication service and device. If an authentication fails, a possibility to register this service manually must be available. This must be carried out on two sides. If no registration has taken place or the attempt fails, the Security manager must determine a standard security level. Also, if authentication attempt fails, a waiting interval, that increases exponentially, must be applied. The device should administrate a list with individual intervals of delay for each communication partner. Due to the small storage capacity of the mobile device the list can contain only the last n units. The user defines the size of the list, that depends also on the memory capacity.

3.7 Concept Realization

Now we present a first realization of the concept shown in section 3.2. The idea is to use an existing mobile device and expand it with required additional hardware components (Fig. 2). For this we use a PDA supporting PCMCIA cards. The PDA itself stands for the processor module and the user interface (key-pad and display). The wireless network interface will be established by a PCMCIA Bluetooth card. The crypto-processor module, the secure memory and the biometric sensor will be integrated on a second PCMCIA card. The advantage of this architecture is that it enables other mobile devices with PCMCIA support to participate on a spontaneous networking environment.

4 Conclusion and Outlook

Wireless mobile devices offer numerous new applications in the environment of the ubiquitous computing. Mobile access to financial transactions, dynamic

information in hotels, airports, offices was some years ago only an unattainable dream. Significant problems of transition to wireless transfer of sensible data are low security features of mobile data communication technologies. The primary goals of this project are implementation of reliable protocols and algorithms for protection of the transmitted data in spontaneously networked mobile devices. Another important issue is the concept of personal identification on a basis of a biometric recognition procedure, that allows to avoid an unauthorized use of the device.

References

1. Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications - the insecurity of 802.11. *University Berkley*, 2001.
2. F. Buchert, W. Bütow, P. Eschoholz, A. Polonsky, F. Quintero, and D. Tavangarian. Ubiquitous Internet and Intranet Access Using WLAN. In *Proceeding of the Workshop on Ubiquitous Computing, International Conference on Parallel Architecture and Compilation Techniques, Philadelphia*, 2000.
3. Bluetooth Consortium. Specification of the Bluetooth System Version 1.0B - Core. http://www.bluetooth.com, 2000.
4. The Infrared Data Association IrDA. http://www.irda.org.
5. M. Jakobsson and S. Wetzel. Security Weaknesses in Bluetooth. *RSA Conference 2001*, 2001.
6. Nico Maibaum and Clemens Cap. Javacards as Ubiquitous, Mobile and Multiserve Cards. 1st Pact 2000 Workshop on ubiquitous Computing, Philadelphia, USA, 15.-19. October 2000.
7. Andreas Pfitzmann, Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Trustworthy user devices. In Günter Müller and Kai Rannenberg, editor, *Multilateral Security in Communications*, pages 137–156. Addison-Wesley, 1999.
8. Jan Rabaey and et al. PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking. *IEEE Computer*, pages 42–48, July 2000.
9. M. Schmalisch, H. Ploog, and D. Timmermann. SECOM: Sichere Online Verschlüsselung fuer ISDN Geräte. 35. Sitzung des Arbeitskreises Technische und organisatorische Datenschutzfragen der Konferenz der Datenschutzbeauftragten des Bundes und der Länder, Rostock, September 2000.
10. Bruce Schneier. *Angewandte Kryptographie*. Addison-Wesley, 1998.
11. Dave Suvak. IrDa and Bluetooth: A Complementary Comparison. http://www.palowireless.com/infotooth/download.asp, 2000.

Author Index

Aguilar, Jose 116
d'Amorim, Marcelo 159

Cap, Clemens 175
Chokhani, Vivek S. 146

Deo, Narsingh 91
Deter, Steffen 53

Eichler, Gerald 19

Ferraz, Carlos 159
Fünfstück, Falk 19

Gmelin, Moritz 134
Gupta, Pankaj 91

Haase, Marc 175
Higuchi, Ken 41
Hoang, Dang-Hai 31
Hochin, Teruhisa 41
Horn, Werner 31

Kastner, Wolfgang 67
Kawahara, Hidetatsu 41
Kim, Ki-Chang 103
Kim, Soo Duk 103
Kim, Yoo-Sung 103
Kreuzinger, Jochen 134

Leiss, Ernst 116
Leupold, Markus 67
Lonchamp, Jacques 167
Lukosch, Stephan 79

Mikler, Armin R. 1, 146
Muller, Fabrice 167

Pfeffer, Matthias 134

Reschke, Dietrich 31
Ricciato, Fabio 19
Rohit, V. Robin 126
Roth, Jörg 79

Sampath, D. 126
Sedov, Igor 175
Sohr, Karsten 53

Tarau, Paul 1
Thomas, Anne 19
Timmermann, Dirk 175
Tsetsekas, Charilaos 19
Tsuji, Tatsuo 41
Tyagi, Satyam 1

Ungerer, Theo 134

Winter, Martin 19