DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
REPORT A-2003-3

Middleware Infrastructure for Distributed Mobile Applications

Stefano Campadello

To be presented, with the permission of the Faculty of Science of the University of Helsinki, for public criticism in Auditorium XII, Main Building, on April 12th, 2003, at 10 o'clock.

UNIVERSITY OF HELSINKI
FINLAND

Contact information

Postal address:
Department of Computer Science
P.O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki
Finland

Email address: postmaster@cs.Helsinki.FI

URL: http://www.cs.Helsinki.FI/

Telephone: +358 9 1911

Telefax: +358 9 191 44441

ISSN 1238-8645
ISBN 952-10-0974-8 (paperback)
ISBN 952-10-0975-6 (PDF)
Computing Reviews (1998) Classification: C.2.4,C.2.6,D.2.11,D.2.12
Helsinki 2003
Helsinki University Printing House

Middleware Infrastructure for Distributed Mobile Applications

Stefano Campadello

Department of Computer Science
P.O. Box 26, FIN-00014 University of Helsinki, Finland
Stefano.Campadello@cs.helsinki.fi

PhD Thesis, Series of Publications A, Report A-2003-3
Helsinki, March 2003, xvi+166 pages
ISSN 1238-8645
ISBN 952-10-0974-8 (paperback)
ISBN 952-10-0975-6 (PDF)

Abstract

One of the most exciting new fields in computer science at the beginning of this millennium is represented by Nomadic Computing. This new technology empowers the user to access typically fixed network services from any place. Mobile users access information services regardless of their physical location or movement behavior and independently of temporal factors. A boost to the Nomadic Computing comes from the improvement of wireless data technology, which began with GSM data communication and now is leading to the deployment of UMTS networks.

Thus, wireless technology and networked applications are starting to find a common path to give answers to the needs of Nomadic users. Unfortunately this merge has been mostly a collision rather than a smooth marriage. Applications were downgraded to let them fit in small devices with poor connectivity, and the results have often been disappointing.

This dissertation focuses on these topics. Firstly, a background overview introduces the challenges that mobile distributed applications face and presents an overview of the main protocols and tools existing in distributed computing. The improvement proposed in literature to address the described challenges are also discussed. Then Java RMI is taken as an example and its problems in a wireless context are analyzed. We propose several improvements to it, we show a prototype implementation giving protocol and messaging details and we give a complete performance evaluation. Secondly, we propose a new approach to Nomadic Computing. A new paradigm shift is suggested, where it is no longer the user who has to adapt to the different scenarios that he may find during his “nomadism”, but it is the service that modifies itself to adapt to the current situation. This new way to design services needs a totally new infrastructure, and brings many new research challenges. We describe them, and following the results of the first part of the dissertation, we suggest an architecture to address them.

Computing Reviews (1998) Categories and Subject Descriptors:
C.2.4 Computer-Communication Network: Distributed System
C.2.6 Computer-Communication Network: Internetworking
D.2.11 Software Engineering: Software Architectures
D.2.12 Software Engineering: Interoperability

General Terms:
Design, Experimentation, Performance, Standardization

Additional Key Words and Phrases:
Middleware, Nomadic Computing, Distributed System, Communication, Software Architecture

Acknowledgments

This work has been a long journey, that started a few years ago in a foggy afternoon in my hometown in Italy and finishes in a sunny but cold afternoon in Finland. And like all journeys, it would have not be possible without the help of many persons.

First of all I have to thank Professor Kimmo Raatikainen, my supervisor but also my mentor. With his support and suggestions he made this journey not only possible but also enjoyable. His skill to read my manuscripts while flying around the world still amuses me.

This work has been carried out mainly at the Department of Computer Science of the University of Helsinki. I would like to express my gratitude to all those persons who are making the department an excellent and friendly working environment. I would like to thank Professors Martti Tienari and Timo Alanko, for their support during my first months in a new country and for having introduced me to their projects, and Jukka Paakki for running the department. My appreciation goes also to Oskari Koskimies and Pauli Misikangas for having shared with me many fruitful discussions. A special mention goes to Heikki "Helluli" Helin for his support and valuable comments and suggestions and for having designed the style of this thesis. It has been delightful sharing many travels with him and Heimo Laamanen during my FIPA experience: I will miss those memorable steaks. Thanks also to Marina Kurtén for correcting the language of this dissertation.

My gratitude goes also to my colleagues at Nokia Research Center in Helsinki. I'm especially thankful to Titos Saridakis and Michael Przybilski for their encouragement and pleasant company. I'm obligated to Heikki Saikkonen and Tapio Tallgren from Nokia Corporation for allowing and encouraging me to finalize this work and for providing an excellent research environment.

A special tribute goes to my friends, especially to Francesco Pento and the "laiset" group. Without them the word "free time" would be meaningless.

Above all, I want to thank all my family for their love and support.

Papà, questo lavoro è dedicato a te.

Helsinki, February 2003

Stefano Campadello

"Deus, dona mihi serenitatem accipere res quae non possum mutare, fortitudinem mutare quae possum atque sapientiam differentiam cognoscere"

Contents

List of Figures

List of Tables

List of Programs

Part I

Introduction

Chapter 1

Introduction

Education is an admirable thing, but it is well to remember from time to time that nothing that is worth knowing can be taught.

- Oscar Wilde

1.1 Introduction

One of the most exciting new fields in computer science at the beginning of this millennium is represented by Nomadic Computing. This new technology empowers the user to access typically fixed network services, such as email, calendar information or web pages, from any place. Mobile users access information services regardless of their physical location or movement behavior and independently of temporal factors.

A boost to the Nomadic Computing comes from the improvement of wireless technology. During the last decade GSM [85] has introduced a data communication [31] that, even if of limited capabilities, is massively available. New protocols with increased performance have followed, such as High Speed Circuit Switched Data (HSCSD) [42, 44] and General Packet Radio Service (GPRS) [43], and Universal Mobile Telecommunications System (UMTS) [26, 94] is expected to be delivered soon.

Thus, wireless technology and networked applications are starting to find a common path to give answers to the needs of Nomadic users. Unfortunately this merger has been mostly a collision rather than a smooth marriage. Applications were downgraded to let them fit in small devices with poor connectivity, and the results have often been disappointing.

Another exciting challenge in this scenario goes under the name of *Interoperability*. Since the computer revolution has begun to change our life, different programming languages, protocols and hardware devices have been designed, implemented and used, making the computer-world lively but chaotic.

Maintaining interoperable devices in typical nomadic computing environments is a task not yet faced by the computer community. It is true that much work has been done so far to improve the performance of TCP/IP communication over wireless links, for example, as we will discuss later in this dissertation, but what is missing is a systematic review of all the layers that compound an application *in a nomadic environment*.

1.2 Motivation

This work will try to fill this lack, at least to some extent. The main goal of this dissertation is to convince the reader that careful design is needed if the word "wireless" is part of the game, whenever the subject is an application, a communication protocol or a middleware component (Figure 1.1). For this purpose we will start describing the different challenges that wireless and mobile computing pose in each of these layers. We will present some of the solutions proposed in the literature, and, if the issue has not been addressed already, we suggest our own solutions.

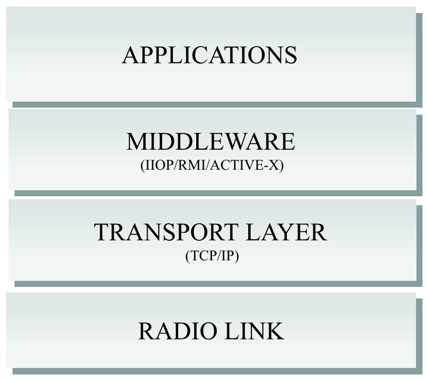


Figure 1.1: The wireless layers

Later on this dissertation becomes more visionary, and tries to suggest a new approach to Nomadic Computing. A new paradigm shift is introduced, where it is no longer the user who has to adapt to the different scenarios that he may find during his “nomadism”, but it is the service that modifies itself to adapt to the current situation. This new way to design services needs a totally new infrastructure, and brings many new research challenges. We will describe them, and following the results of the first part of the dissertation, we will suggest an architecture to address them.

1.3 Overview of the Dissertation

The rest of this dissertation is divided into five parts. The following part contains Chapters 2 and 3 and presents a background overview. Chapter 2 introduces the challenges that mobile distributed applications face, while Chapter 3 gives an overview of the main protocols and tools existing in Distributed Computing.

Part three (Chapters 4,5,6) addresses the infrastructure for Nomadic Applications. Chapter 4 gives an overview of the improvement proposed in literature to address the challenges described in the previous part. Chapter 5 is one of the main contributions of this thesis. It takes Java RMI as an example and analyzes its problems in a wireless context, proposes several improvements, shows a prototype implementation giving protocol and messaging details and gives a complete performance evaluation. Chapter 6 presents new architectures devoted to Nomadic Computing and introduces the concepts of Pervasive Computing.

The fourth part presents a new paradigm to allow a dynamic adaptation of applications to a nomadic environment. Chapter 7 gives the reader the motivation for this new approach. Chapter 8 introduces the use of Agent technology in Personal Mobility scenarios, while Chapter 9 enters into details of the proposed new architecture, describing a proof of concept, as well.

The final part of the thesis (Chapter 10) draws the conclusions and makes the final remarks.

1.4 Research History

This thesis, even if written in the form of a monograph, consists of several conceptual ideas that the author has developed during his research and presented in international conferences or fora. Preliminary contributions have been shared with the Mowgli project (see Section 4.2.3) at the University of Helsinki and within the Dolmen project (see Section 4.4.1) where the author implemented part of the MDBR.

Influenced by this knowledge the author has designed, implemented and evaluated Wireless Java RMI, here introduced in Chapter 5 but already presented, in less details, in [21, 22]. This work has been carried out in the framework of the Monads project (see Par. 4.2.2.1). Other results obtained by the author from the Monads project have been presented in [20].

The interest of the author in the role of agents in a nomadic environment is influenced by his participation in the FIPA standard committees, especially the ones addressing the adaptation of FIPA specifications to the nomadic environment. The contributions of the author have been incorporated in the resulting specifications [37, 38, 39]. The work done in FIPA has given motivation for two other articles written by the author, [62, 50]. Nevertheless, all these contributions regarding agents are not fully presented in this dissertation. For a complete presentation we refer to [49].

The author's ideas presented in the fourth part get their roots from these previous works. Chapter 8 has been essentially presented in [23] while an initial version of Chapter 9 has been published in [19].

1.5 Mobile User and Mobile Code

In this dissertation we will focus on the nomadic users and their needs. In this respect, we consider the applications they use as part of their nomadism, being them situated, for example, on mobile devices.

A different approach is to consider a mobile application as an application where the code moves from site to site, mainly in a wireline environment, regardless of the position of the end user. Examples of this approach are, for instance, platforms for mobile agents. There are many conferences, books and fora devoted to mobile agents, and even the Java platform, by mean of Enterprise Java Beans (EJB) [73] or Jini (see Section 6.3.3), considers mobile code.

In this dissertation we keep our focus on Nomadic applications rather than on mobile code, but in our belief this two visions will soon merge, as the work done in FIPA, as cited before, proves. It is not by chance that the architecture we introduce in the last chapters of this dissertation uses mobile code.

Part II

Background Overview

Chapter 2

The Challenges in Mobile Distributed Applications

Okay, Houston, we've had a problem here.
- John L. Swigert, Command module pilot, Apollo XIII

2.1 Introduction

The concept of *Nomadic Computing* [9, 59], comes from a user desire: to have access to the preferred computer service "Anytime, Anywhere". The proliferation of computing devices such as laptops or palmtops and their increase of computational power and usability has given the users the possibility to move around the world bringing with them their own equipment. The demand to be able not only to carry the devices, but also to use them during the move was a natural consequence.

As Kleirock says, *"The essence of a nomadic environment is to automatically adjust all aspect of user's computing, communications, and storage functionality in a transparent and integrated fashion"* [58].

The evolution of wireless technology has been as rapid, and the meeting between telecommunications and computer science has been fast and rich, but not without pain. Unfortunately, in fact, the usual assumption that engineers and computer scientists had before was that networking meant to be "always connected". And this is not the case anymore. Moving from the office to home, reading e-mail in an airport lounge or writing a report in a train cannot be considered as an exceptional case nowadays. Thus, in networking, the disconnected mode cannot be considered a failure any longer, but one of the possible user scenarios.

The request for a *transparent* adaptation of these changes involves location transparency, communication device independence, adaptation to communication bandwidth variation, and mobility transparency. The applications willing to address the Nomadic user have to be adaptive, evolving with the given quality of service and computing capabilities at any time.

It is clear then that the conjunction between wireless technology and computer science opens many new bright scenarios, but also leads to numerous new challenges. They are analyzed in the following sections.

2.2 The Challenges in Mobile Computing

Mobile Computing can be defined as a paradigm of computing in which users who carry portable devices have access to information services through a shared infrastructure, regardless of their physical location or movement behavior. There are many examples of scenarios that exploit the concepts of Mobile Computing:

- the possibility to use a laptop computer during travel with the ability to upload the work done once disembarked from the aircraft
- the possibility to follow financial information independently from the actual location, thus being able to take the right actions on time. For instance, selling or buying stocks requires perfect timing
- being able to instantiate a communication network during a disaster recovery mission. For example, fulfill the need to coordinate the search of survivors after a major earthquake
- the possibility to access Internet services and information while on the move. For instance, to be able to read e-mail at any time or to obtain the address of the restaurant in a foreign town (and maybe the description of the fastest way to reach the place).

It could seem that the potential of this new paradigm is limited only by the imagination on the service designers, but not everything is ready for the revolution. Most of the services the mobile users want to access are designed to be static and accessed by static users. But, while traditional techniques are based on the assumption that the location of hosts in distributed systems does not change and the connection among hosts does not change either during the computation, in a mobile environment these assumptions are rarely appropriate. Thus building the infrastructure needed to enable Mobile Computing leads to many challenges [40], as described in the following sections.

2.2.1 Communication Issues

Mobile computers and devices require a wireless network to access the network while on the move, and wireless communication faces more obstacles than wired communication because the surrounding environment interacts with the radio signal, introducing echoes, noises or even blocking it. As a result, wireless communication is characterized by lower bandwidths, higher error rates, and frequent disconnections. These factors can furthermore increase communication latencies resulting from retransmissions, transmission timeout delays and error-control protocols ([2, 53]). In addition, the wireless connection can degrade or be lost while the users move, and if the number of mobile users in the wireless network is elevated, the network capacity can be overloaded at the service's expense.

2.2.1.1 Low Bandwidth

Wireless networks deliver lower bandwidth than wired networks and are two or three orders of magnitude slower. Sending a long file to the wireless network, for instance, could require very long time, and the chance to experience a network failure increases with time. A graphical user interface can act in a bizarre way if accessed remotely due to the length of the response-time, becoming useless to the user.

2.2.1.2 High Bandwidth Variability

As a consequence, mobile devices experience much greater variation in network bandwidth than traditional computers. The bandwidth can increase (or decrease) of one to four orders of magnitude, depending on whether the system is plugged in a wired network or in a wireless domain. A video conference, for example, is possible while the network is wired, but cannot sustain the same quality of service when the system is unplugged and the transmission goes over a wireless network.

2.2.1.3 Disconnections

Computer systems depend heavily on a network connection and they may not function properly if the network experiences failures. For example, distributed file systems may lock up waiting for a remote server to allow access, or an application can simply fail if it does not obtain any answer from the network. Once the network disconnection ends, another major problem is the recovery phase. The disconnected terminal must be reintegrated in the network, and possible conflicts due, for example, to stale data, must be resolved.

2.2.2 Mobility Issues

The ability to change locations while connected to a network increase the volatility of some information. Data considered static for stationary systems becomes dynamic in Mobile Computing. For example, a stationary computer could be configured statically to use the printer in the next room, while a mobile computer needs a mechanism to find the available printers and to select one.

Mobility introduces several problems: A mobile computer's network address changes dynamically, its current location affects configuration parameters, and its communication bandwidth varies according to its position [8, 17, 102].

2.2.2.1 Address Migration

In Mobile Computing, the physical location of a mobile unit does not determine its network address. In fact, while the users move, their mobile devices will use different access points to the network, and thus different network address. In classical networking, once an address for a computer is known to the system, it is cached with long expiration time. This issue poses a challenge in Mobile Computing since to communicate with a mobile computer, messages must be sent to the most recent address.

2.2.2.2 Location-dependent Information

Since traditional computers do not move, information that depends on location, such as local printers and local conventions (as local currency), is typically configured statically. The challenge is to find these pieces of information automatically and to configure the device conforming to the actual location.

2.2.3 Devices Issues

Classical systems and applications are usually designed for static devices. Mobile devices need to be small, light, durable and being operational under wide environmental conditions. They require minimal power usage for a long battery life.

2.2.3.1 Power Limitation

Energy supply is the major bottleneck for mobile wireless computers. Batteries are the largest single source of weight in a portable computer, and users desire to carry light devices. On the other hand, longer battery life is another feature requested by mobile users. Thus, energy efficiency is a necessity, and a challenge, both at the level of hardware, and software.

2.2.3.2 User Interface and Display Issues

Size constraints on a mobile device force to use small user interfaces. Small display size is a serious problem, especially for users who want to access remote information services, such as those provided on the Internet [64]. But input devices need also to be redesigned, since the shortage of surface area in modern portable devices suggests to sacrifice large input devices, like keyboards, in favor of analog input, such as pen-based interfaces. This redesign poses new challenges, especially in hand-writing recognitions.

2.2.4 Security Issues

Mobile Computing imposes special security requirements, particularly with regard to identification and certification [7, 47, 63].

Because the device is not the person, technology can hide the identities of the communication parties. Ensuring that the persons at the other end of the communication channel are who they assert they are is critical, for example, in billing. Privacy is at risk when utilizing insecure channels, as is possible in wireless communication. Applications are at risk when new data is downloaded from an untrusted site and data is at risk when downloaded to an untrusted side. In general, Mobile Computing raises all the security challenges that can apply to Computer Science.

Chapter 3

Middleware for Distributed Computing

And how is education supposed to make me feel smarter? Besides, every time I learn something new, it pushes some old stuff out of my brain.

- Homer Simpson

3.1 Introduction

In the previous chapter we described the major challenges that designers of applications for a nomadic user have to face. But there is also another level of interaction in modern computing that is affected by mobility and a wireless environment: Distributed Computing.

The rise of networked workstations and fall of the centralized mainframe has been the most dramatic change in the last two decades of information technology. As the number of computer networks has increased, the concept of distributing services among multiple computers has become increasingly possible and desirable. This new concept has been widely implemented, and modern operating systems, like Unix, embed distributed services inside the system.

In this chapter we focus our attention on a subset of all the vast field of distributed computing and distributed systems, which deals with the management of remote procedure calls or objects.

The ability to access remote resources as they were local is of particular interest to Nomadic users, since this paradigm could automatically hide the communication details to the developers and the users. Unfortunately, as we will see, often the implementations of this concept do not fit well in wireless and mobile environment.

In the following sections we illustrate the main protocols in use and the reasons why they fail in providing useful service to Nomadic users.

3.2 Remote Procedure Calls

Remote Procedure Calls (RPC) [93] is a paradigm for providing communication across a network between programs written in a high-level language. RPC implements a logical client-to-server communication system designed specifically for the support of network applications. The idea of RPC has been discussed in the literature since 1976 [103]. Nelson's doctoral dissertation [77] presents extensively the design possibilities for an RPC system.

The primary purpose of RPC is to make distributed computing easy. It was previously observed that the construction of communicating programs was a very difficult task, even for researchers with a good knowledge of the distributed computing concept. Since procedure calls are a well-known and a well-understood mechanism for transfer of control and data inside a program running on a single machine, it was proposed to extend the same mechanism to transfer data across a network.

The flow of control in a RPC call is depicted in Figure 3.1. The control goes logically through two processes: the caller process and the server process. First, the caller process sends a call message that includes the procedure parameters to the server process. Then the caller process waits for a reply message (synchronous call). Next, a process on the server side, which is dormant until the arrival of the call message, extracts the procedure parameters, computes the results, and sends a reply message. The server waits for the next call message. Finally, a process on the caller receives the reply message, extracts the results of the procedure, and the caller resumes execution.

RPC presumes the existence of a low-level transport protocol, such as TCP/IP or UDP, for carrying the message data between communicating programs. RPC provides an authentication process that identifies the server and client to each other and includes a slot for the authentication

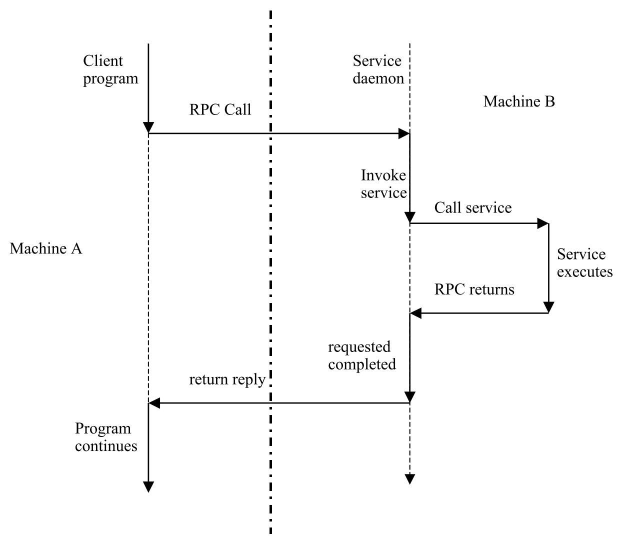


Figure 3.1: Network communication with the Remote Procedure Call.

parameters on every remote procedure call so that the caller can identify itself to the server. The client package generates and returns authentication parameters.

The RPC interface is generally used to communicate between processes on different workstations in a network. However, RPC works just as well for communication between different processes on the same workstation.

The use of the RPC protocol is declining, since its main implementations are not written in object-oriented programming languages, thus the interest in a wireless or mobile version of RPC is low. In the next session we describe Java RMI, which is seen as the modern and object oriented version of RPC.

3.3 Java RMI

Remote Method Invocation (RMI) [69] is the object-oriented version of RPC. RMI is essentially the same concept that allows the programmer to transparently invoke methods on objects that reside on another computer. In this way, the object-oriented paradigm is preserved in distributed computing.

Java RMI was designed to simplify the communication between two objects in different Java Virtual Machines (VM) by allowing transparent calls to methods in remote virtual machines. Figure 3.2 depicts the protocol stack in the Java RMI communication.

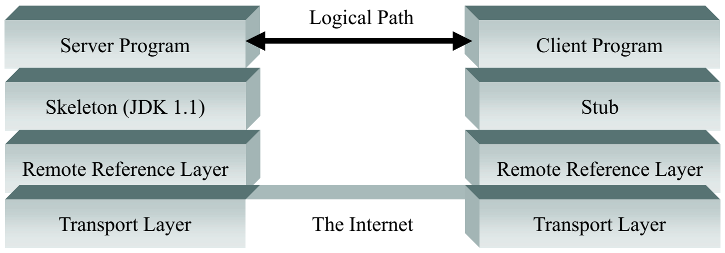


Figure 3.2: Java RMI Layers

Once a reference of a remote object is obtained, it is possible to call methods of that object in the same way as methods of local objects. Since the remote object resides in a different virtual machine, an RMI *Registry* is needed to manage remote references. When an RMI server wants to make its local methods available to remote objects, it registers those methods to the local registry. A remote object connects to the remote registry, which listens to a well-known socket, and obtains a remote reference.

Java RMI is built on top of a *transport layer*, which provides abstract RMI connections built on top of TCP connections. When an RMI connection is opened, the transport layer either opens a new TCP connection, or reuses an existing one if a free one is available. If the reused connection has been idle for more than the time of a round-trip, the transport layer first sends a ping packet to make sure the connection is still working. Once an acknowledgment for the ping packet is received, the new RMI connection is established. If a TCP connection has not been used by any RMI connections for a while, it is closed.

The general Java RMI architecture is depicted in Figure 3.3. First a server creates a remote object and registers it to the local Registry (1). The client then connects to the remote Registry (2) and obtains the remote reference. At this point, a stub of the remote object is transferred from the remote virtual machine to the client virtual machine. This happens only if the stub is not yet present in the local VM. When the client (3) invokes a method at a remote object, the method is actually invoked at the local stub. The stub marshals the parameters and sends a message (4) to the associated skele-

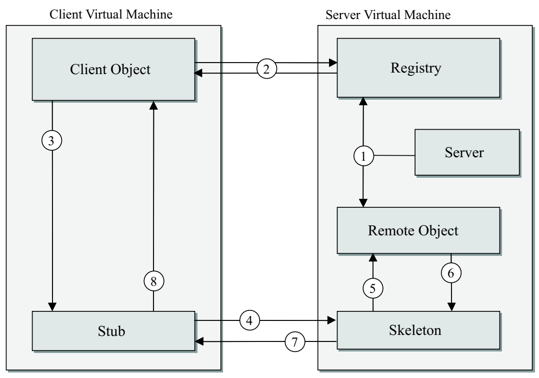


Figure 3.3: Java RMI Protocol

ton on the server side. The skeleton unmarshals the parameters and invokes the appropriate method (5). The remote object executes the method and passes the return value back to the skeleton (6), which marshals it and sends a message to the associated stub on the client side (7). Finally the stub unmarshals the return value and passes it to the client (8).

3.3.1 RMI in Nomadic Computing

As explained in more detail in section 5.1, RMI is not suited to be used in a wireless environment, thus it is not of great use in Nomadic and Mobile Computing. Getting a single reference to a remote object in a GSM data environment requires between 8 and 20 seconds, depending on the version of the Java virtual machine. The main reasons for this fact lay in the way the protocol is implemented, with a heavy use of underlying TCP connections. And, as we will see, TCP behaves poorly in wireless communication. Other reasons are directly dependent on the protocol (and thus independent from the implementation) like the use of Serialization and Distributed Garbage Collection. The second point is that RMI in not designed to be used in mobile hosts: If the client's host changes Internet address the protocol fails. In section 5.1 we suggest a solution to these problems and we present the results of our prototype implementation.

3.4 OMG CORBA

The Common Object Request Broker Architecture (CORBA) [80] specifies a system which provides interoperability between objects in a heterogeneous, distributed environment and in a way transparent to the programmer. This open distributed object computing infrastructure is being standardized by the Object Management Group (OMG) [82], that is, a non-profit consortium created in 1989 with the purpose of promoting theory and practice of object technology in distributed computing systems.

The idea behind CORBA is essentially the same as the one behind RMI, but the fundamental difference is that while Java RMI is meant to enable interoperability between Java platforms, CORBA is programming-language independent. Figure 3.4 illustrate the essential components of the CORBA Architecture.

In this model clients request services from objects through a well-defined interface. This interface is specified in OMG IDL (Interface Definition Language). A client accesses an object by issuing a request to the object. The request is an event, and it carries information including an operation, the object reference of the service provider, and actual parameters (if any). The central component of CORBA is the Object Request Broker (ORB). It encompasses all of the communication infrastructure necessary to identify and locate objects, handle connection management and deliver data. The ORB Core is the most crucial part of the Object Request Broker since it is responsible for communication of requests.

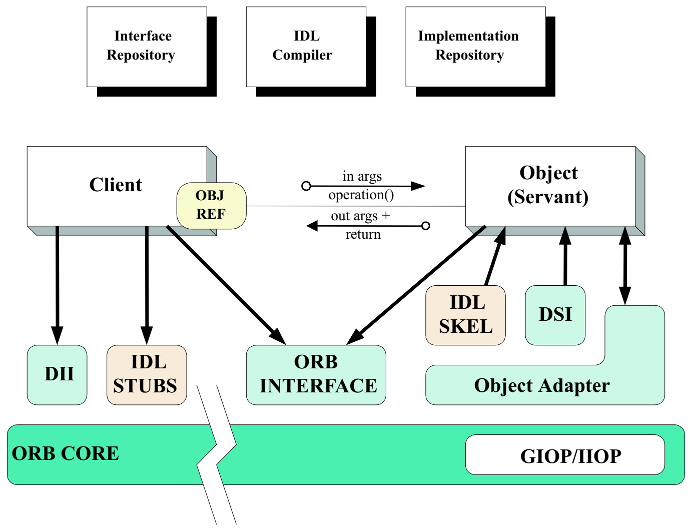


Figure 3.4: CORBA Architecture

Going into more details, the essential components in the CORBA architecture are:

Object – This is a CORBA programming entity that consists of an identity, an interface, and an implementation. It is the place where the client requests the service.

Servant – This is an implementation programming language entity that defines the operations that support a CORBA IDL interface. Servants can be written in a variety of languages, including C, C++, Java, Smalltalk, and Ada.

Client – This is the program entity that invokes an operation on an object implementation. Accessing the services of a remote object is transparent to the caller.

Object Request Broker (ORB) – The ORB provides a mechanism for transparently communicating client requests to target object implementations. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.

CORBA IDL stubs and skeletons – CORBA IDL stubs and skeletons serve as the "glue" between the client and server applications, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler.

Dynamic Invocation Interface (DII) – This interface allows a client to directly access the underlying request mechanisms provided by an ORB. Applications use the DII to dynamically issue requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs (which only allow RPC-style requests), the DII also allows clients to make non-blocking deferred synchronous (separate send and receive operations) and one-way (send-only) calls.

Dynamic Skeleton Interface (DSI) – This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.

Object Adapter – This element assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects).

3.4.1 CORBA in Nomadic Computing

Also CORBA, as it exists at the time of writing, is less than perfectly suited for wireless access. Certain assumptions have been built into CORBA that are not valid for wireless networks and which create difficulties for applications using wireless access. The fundamental challenges for CORBA in wireless and mobile environments include reliability of transport, which is much lower in wireless networks than in fixed networks, and mobility of terminals, which implies that the terminal may change its point-of-presence in the network.

We return to these issues in Section 4.4.2.

3.5 Microsoft COM and DCOM

Component Object Model (COM) [25] developed by the Microsoft Corporation, provides a support for interoperability and re-usability between distributed objects. Distributed COM (DCOM) [28] is an extension of COM allowing interaction between objects running in different machines. DCOM is completely integrated in COM, and they can be considered a single technology.

COM allows a *binary* compatibility between the client and the object, written in arbitrary languages, through the use of interfaces in a similar manner as Java RMI. COM objects and interfaces are specified using Microsoft Interface Definition Language (IDL).

There are three ways in which a client can access COM objects:

1. In-process server (Figure 3.5). Both the client and the server execute in the same process. The client calls methods in the component without any overhead.

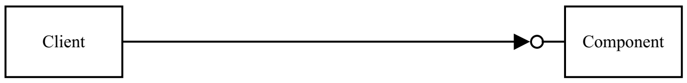


Figure 3.5: DCOM objects in the same process

2. Local Object Proxy (Figure 3.6). The client and the component can access a server running in a different process executing in the same machine.

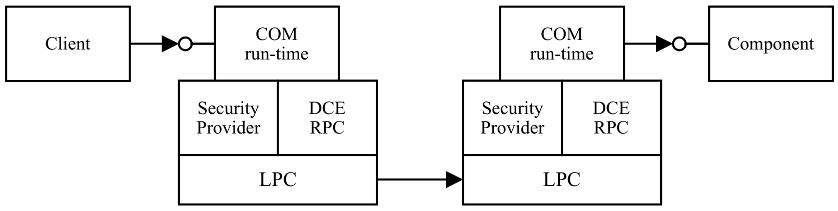


Figure 3.6: DCOM objects in the different processes

3. Remote Object Proxy (Figure 3.7). When client and component reside on different machines, DCOM simply replaces the local interprocess communication with a network protocol.

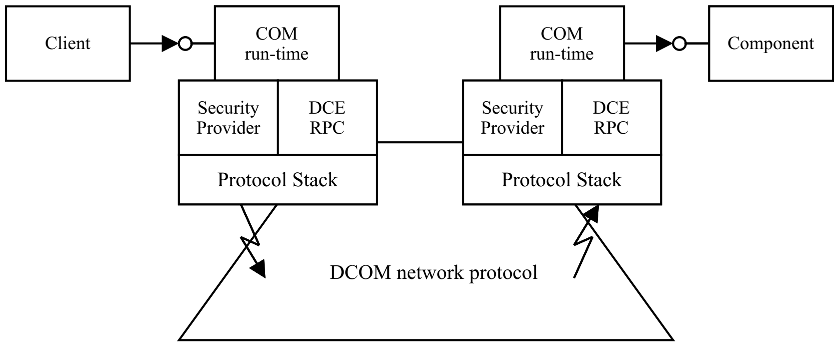


Figure 3.7: DCOM objects in different machines

When the client is separated from the server, the data must be marshalled. As in Java RMI and CORBA, marshalling is accomplished by a *proxy* and a *stub* object that handle the cross–process communication. All COM objects are registered in a component database. The clients obtain the address of the methods implemented by the server through a simple table of method addresses called *vtable*.

3.5.1 DCOM for Nomadic Users

Since COM and DCOM are based on a native binary format, the components implementing these specifications are not platform-independent. Furthermore, COM and DCOM are best supported on Microsoft Windows platforms. Even if support for other platforms is on the way, the technology will be primarily Windows (and thus proprietary) based. For these reasons we do not foresee a practical use of COM/DCOM in an open and dynamic environment like Nomadic Computing, where proprietary solutions can represent impenetrable barriers to mobility.

3.6 WAP

The Wireless Application Protocol (WAP) [100, 101] is an attempt to define an open standard for how content from the Internet is filtered for mobile communications. From its very beginning WAP has been designed for Mobile and Nomadic Computing, even if the first target is to provide Internet access to wireless devices such as digital cellular phones, pagers and PDAs.

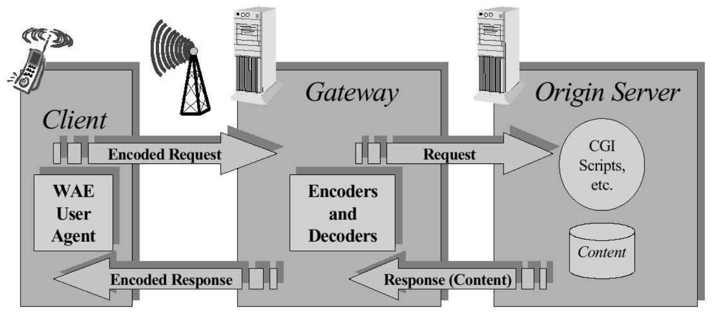


Figure 3.8: WAP Programming Model

The Wireless Application Protocol takes a client-server approach. It incorporates a relatively simple *microbrowser* into the mobile device, requiring only limited resources on it. This makes WAP suitable for thin clients and early smart phones. WAP puts the intelligence in the WAP Gateways while adding just a microbrowser to the mobile devices themselves (Figure 3.8). WAP utilizes proxy technology to connect between the wireless domain and the Internet. In this way content and applications can be hosted on standard web servers.

WAP has a layered architecture as shown in the diagram below:

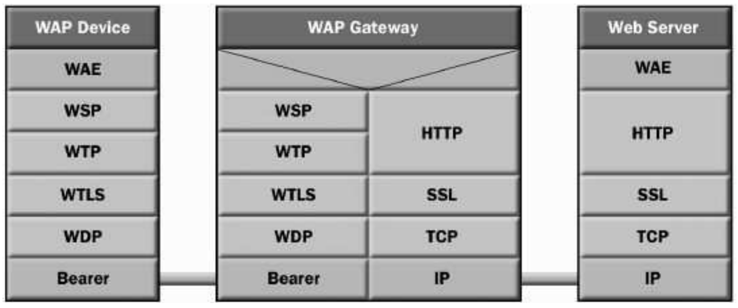


Figure 3.9: WAP Architecture

3.6.1 Wireless Application Environment (WAE)

The application layer of WAP is a compound of The Wireless Application Environment (WAE) and the Wireless Telephony Application (WTA). They allow the developer to use specific formats and services created and optimized for interacting with devices with limited capabilities. WAE formally specifies just the formats such as images and text formats that the applications have to be compliant with but says nothing about the client implementations[1]. WTA is a collection of telephony-specific extensions that allows control over the telephone device. WAE contains a lightweight markup language (WML), and a lightweight scripting language (WMLScript).

3.6.2 Wireless Session Protocol (WSP)

The WSP layer provides a lightweight session layer to allow efficient exchange of data between applications. In particular, WSP supports the efficient operation of a WAP micro-browser running on the client device and communicating over the low-bandwidth and high-latency wireless network. This layer is similar to the existing HTTP 1.1 standard, being semantically equivalent. Typically, the HTTP messages are transformed into WSP messages before being sent over the air. WSP introduces features that address many of the limitations that are inherent to HTTP. WSP establishes a long-lived session between the client and the WAP gateway and it uses an efficient binary encoding instead of plain ASCII text.

[1]Client applications are typically microbrowsers and text message editors.

3.6.3 Wireless Transaction Protocol (WTP)

WTP is situated on top of a datagram service such as User Datagram Protocol (UDP), to provide a simplified protocol suitable for low bandwidth mobile stations. WTP offers three classes of transaction service: unreliable one-way request, reliable one-way request and reliable two-way request respond. It supports delayed acknowledgment and handshake minimization to help reduce the number of messages sent over the wireless link. WTP delays the transmission of data and acknowledgments in the hope to concatenate multiple messages into a single transmission, thus reducing network overhead.

3.6.4 Wireless Transport Layer Security (WTLS)

WTLS is modeled after the Transport Layer Security (TLS) [30] that is a "de facto" standard over the Internet. WTLS provides authentication using certificates optimized so that they demand less bandwidth than the traditional certificates sent over the Internet. This protocol includes data encryption, to prevent third parties from seeing or modifying the data. It is also designed to defend the device against various security attacks, including *reply attacks* and *denial-of-service attacks*.

The WTLS protocol is optional in the WAP stack. A device is not required to support WTLS, and even when it is present its use is optional. This is due to the fact that WTLS raises computation and bandwidth requirements. Being optional it allows WAP to be deployed on devices with minimal resources.

3.6.5 Wireless Datagram Protocol (WDP)

WDP is the bottom layer of the WAP stack. It shields the upper layers from the bearer services provided by the networks, allowing the applications a transparent transmission of data over the different bearers. WDP is also responsible for packet segmenting and reassembling.

3.6.6 WAP 2.0

The activity of the WAP forum has continued during the years, and many changes have been made to the standards to improve interoperability and to support the upgraded or new networks and network bearers that have been introduced, as General Packet Radio Service (GPRS) [43], High-Speed Circuit-Switched Data (HSCSD) [42, 44] and UMTS [26, 94]. In 2001 the forum released a whole new body of specifications, the 2.0 version. This new version includes an additional WAP stack, as depicted in Figure 3.10.

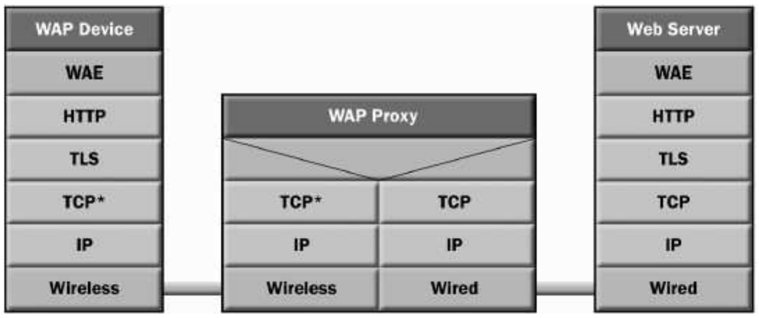


Figure 3.10: WAP Architecture 2.0

This stack supports Internet protocols when IP connectivity is available to the mobile device. This support has been motivated by the emergence of high-speed wireless networks that provide IP support directly to the wireless devices. Backward compatibility with the 1.x protocol stack is assured too.

A further enhancement is the introduction of XHTML Mobile Profile markup language (XHTMLMP). This markup language extends the basic profile of XHTML as defined by the W3C consortium [98].

Part III

Infrastructure for Nomadic Applications

Chapter 4

Enhancing Infrastructure for Nomadic Applications

Computers don't make errors—What they do they do on purpose.
- Dale Gribble

4.1 Introduction

The limitations and constraints presented in Section 2 represent a challenge to a wide adoption of mobile data services. To overcome this problem there have been several researches addressing issues of mobile systems and applications, especially for the purpose of mobile information access. These solutions attack the problem from different perspectives. A group of them try to enhance the communication layer directly, fighting the problems where they come from. Another group focuses their attention to the middleware, the closest place to the applications. Furthermore, another group tries to add mobility to existing protocols. In this chapter we describe these solutions.

4.2 Enhancing the Communication Layer

4.2.1 Improving TCP over Wireless Links

The Internet Engineering Task Force (IETF) published a document [74] suggesting the implementers how to improve the behavior of TCP so that it would also satisfy to the users of what they call the *Long Thin Networks*. The name derives from the observation that wireless networks are *Long* networks because their round-trip time is quite long, and *Thin* because their bandwidth is usually low compared with the one of wireline networks[1]. The paper identifies the following problems in TCP over Long Thin Networks:

Slow Start and Congestion Avoidance whenever TCP's retransmission timer expires, the sender assumes that the network is congested and invokes slow start congestion control. In a wireless environment retransmissions are mostly triggered by corruption, thus the slow start is wrongly requested.

Delayed ACKs the sender increases the dimension of its window depending on the number of ACKs received. The number, of course, is dependent on the round-trip time between sender and receiver, which implies that TCP's adaptation is correspondingly slower than on networks with shorter delays.

Three-way Handshake TCP starts a *three-way handshake* whenever a new connection is set up. Data transfer is only possible after this phase has completed successfully. On networks with long latency and for short transactions this handshake wastes valuable time.

Many proposals have been made to modify or eliminate slow start in long latency environments. Solutions vary from using a larger initial window during the slow start [5] to counting the data acknowledged and not the number of ACKs [4]. Other proposals include a change in the actual slow start protocol: While adding ACKs, they suggest changing the spacing of the acknowledges or to delay duplicate ACKs.

Another issue is the length of the headers in TCP/IP: Because Long Thin Networks are bandwidth-constrained, compressing every byte out of over-the-air segments is worthwhile [24, 29, 55, 34]. Compressing the header improves the interactive response time, allows using small packets

[1]Satellite links, not considered here, make *Long Fat Networks* (LFN), since they may have high bandwidth. Satellite networks may often show a delay*bandwidth product above 64 KBytes. For a Wireless LAN a typical round-trip time may be around 500 ms, and the sustained bandwidth is about 24 Kbps. This yields a delay*bandwidth product roughly equal to only 1.5 KBytes.

for bulk data with improved efficiency and decreases header overhead[2]. Other proposals [92] suggest to compress the IP payload, but, in general, compression made at the application level can outperform these solutions.

SNOOP Berkeley's Snoop protocol [10] is a hybrid scheme mixing link-layer reliability mechanisms with the split connection approach. It is an improvement over split TCP approaches in that end-to-end semantics are retained, meaning that the Snoop protocol does not break the TCP connection. Snoop does two things:

1. Locally (on the wireless link) it retransmits lost packets, instead of allowing TCP to do so end-to-end.
2. It suppresses the duplicate ACKs on their way from the receiver back to the sender, thus avoiding fast retransmit and congestion avoidance at the latter.

The Snoop protocol is designed to avoid unnecessary fast retransmits when packets are received unsorted and duplicated acknowledgments are received by the sender. The optimization is made in the intermediate node between the sender and the wireless receiver, so this solution works well only when such an intermediate node exists. Another problem with SNOOP is that it does not work if the IP traffic is encrypted and the intermediate node does not share the security association between the mobile device and the sender.

4.2.2 Improving the Client-Server Paradigm

Moving from the TCP/IP level toward the application layer we recognize the need for adaptation to the rapid changes in network conditions. Satyanarayanan [88] identifies two extremes in the strategies for adaptation (Figure 4.1). The range is delimited by two extremes.

At one extreme, adaptation is entirely the responsibility of individual applications. This approach, called *laissez-faire adaptation*, avoids the need for system support. The other extreme, called *application-transparent adaptation*, places the entire responsibility for adaptation on the system.

[2]For a common TCP segment size of 512 the header overhead of IPv4/TCP within a Mobile IP tunnel can decrease from 11.7 to less than 1 per cent.

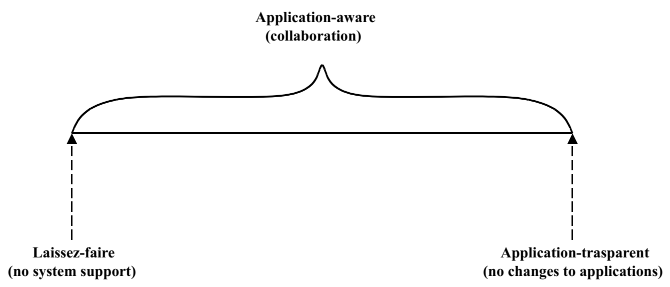


Figure 4.1: Range of adaptation strategies

Between these two extremes there exists a multitude of mixed solutions that are referred to as *application-aware adaptation*. The laissez-fair approach lacks a central element to control and limit the resource requested by the different applications. The second approach is more attractive, since it preserves the compatibility with existing applications, but there may be situations where the adaptation wholly controlled by the system is inadequate or counterproductive. A typical case of application-transparent adaptation is to use proxies to perform adaptation on behalf of applications. We used this kind of adaptation for the architecture presented in Chapter 5.

4.2.2.1 Application-Transparent Adaptation

Examples of application-transparent adaptation addressing adaptation for mobile file system applications are Little Work [52] and the Rover Toolkit [57, 86]. In these examples, a local proxy runs on the mobile host and provides an interface for regular server services to the applications. The proxy attempts to mitigate any adverse effects of mobile environments.

Web proxies enable Web browsing applications to function over wireless links without imposing changes on browsers and servers. Web proxies can be used to prefetch and cache Web pages to the mobile client's machine, to compress and transform image pages for transmission over low bandwidth links, and to support disconnected and asynchronous browsing operations.

Below, other Application-Transparent proposals are briefly summarized.

CODA The most famous example of Application-Transparent Adaptation is the CODA file system [90]. To provide adaptability the system uses a file system proxy to make existing applications work with no modification (Figure 4.2).

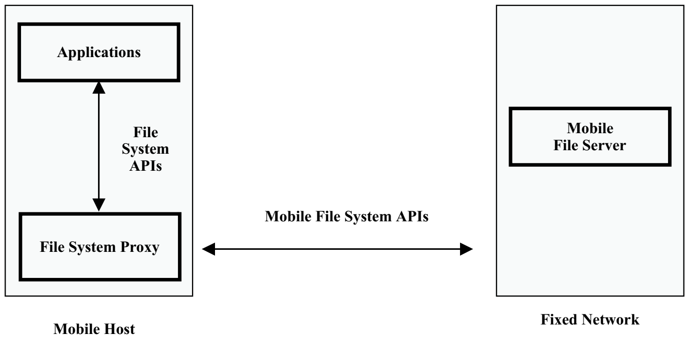


Figure 4.2: The CODA file system

The proxy records all updates to the file system during disconnection and synchronizes on reconnection. Automatic mechanisms for conflict resolution using optimistic concurrency control are provided for directories and files through the proxy and the file server. The file system proxy in CODA allows disconnected operations, optimistic replication and has a support for weak connectivity.

WebExpress WebExpress [48] uses a Web Proxy approach to intercept and control communications over the wireless link for the purposes of reducing traffic volume and optimizing the communication protocol to reduce latency. A proxy approach is also used in the Mowgli project (see section 4.2.3).

The approach adopted by these solutions does not require changing existing applications for running in mobile environments. However, it could sacrifice functionality and perhaps performance. As applications are shielded from dealing with mobility, it might be very hard for the system to make adaptation decisions that meet the needs of different and diverse applications. As a result, it may have to require some manual intervention by the user (for example having the user indicate which data to prefetch onto the mobile device) to make applications run smoothly. Such user-administered manual actions could be less agile to adapt to the changing environment.

Monads A research project carried out by the University of Helsinki and called MONADS [20, 72] addresses this problem. The project examines adaptation agents for nomadic users designing and implementing a software architecture based on software agents (Figure 4.3) . The Monads architecture is based on the Mowgli communications architecture (see Section 4.2.3), that takes care of data transmission issues in wireless environments. In addition, the project made use of existing solutions, such as FIPA [36] specifications as far as possible. The principal idea in the Monads project has been that nomadic applications are offered information about the future quality of the connection, and they are supposed to adjust their behavior to meet the forthcoming situation.

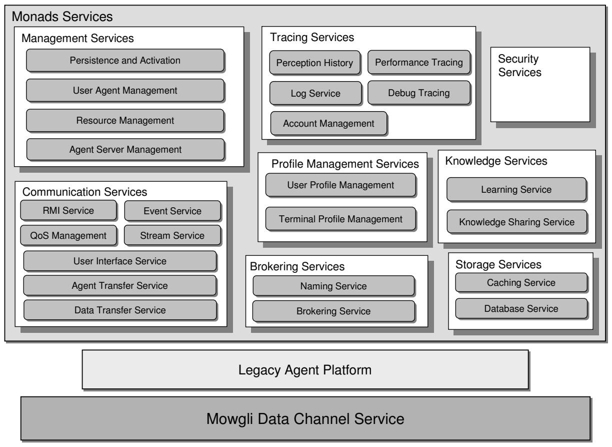


Figure 4.3: The Monads architecture

In the Monads philosophy, software systems that are to be used in wireless environments should be able to adapt to sudden changes in the quality of data transmission over wireless connections. As a minimum, a system should detect when current data transmission tasks may not be completed any longer in a reasonable amount of time due to temporary changes in the QoS. More sophisticated systems could try to adapt to the current QoS by using special data filtering and compression methods, and to refuse to accept requests that cannot be fulfilled within a certain time limit. In Monads, adaptation is mainly achieved by learning; the architecture's main focus has been on learning to predict QoS [70].

4.2.2.2 Application-Aware Adaptation

Application-aware adaptation allows applications to react to the mobile resource changes. One way to realize the application-aware adaptation is through the collaboration between the system and individual applications. The system monitors resource levels, notifies applications of relevant changes, and enforces resource allocation decisions. Each application independently decides how best to adapt when notified. Examples of this approach are the Odyssey Project[79], the BARWAN Project [16] and the Prayer System [14]

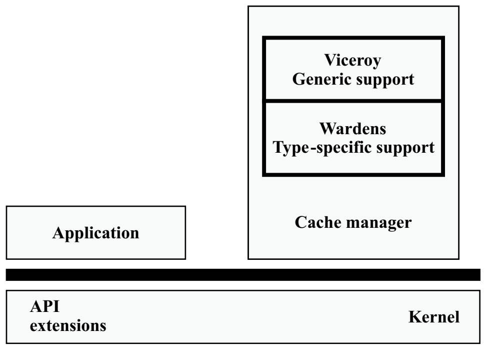


Figure 4.4: The Odyssey client architecture

Odyssey The application-aware adaptation in Odyssey (see Figure 4.4) is performed through the use of type-specific operations between the system and applications. The type-awareness is incorporated into both the system for efficient resource usage and the applications for differential handling of data types. The system-level knowledge of data types facilitates the optimization of the resource usage for different and diverse applications. For example, the size distribution and consistency requirements of data from an NFS server differ substantially from those of relational database records. Image data may be highly compressible using one algorithm, but not another. Video data can be efficiently shipped using a streaming protocol that drops rather than retransmits lost data; in contrast, only reliable transmissions are acceptable for file or database updates. Odyssey incorporates type-awareness via specialized code components called *wardens*. A warden encapsulates the system-level support at a client to effectively manage a data type. To fully support a new data type, an appropriate warden has to be written and incorporated into Odyssey at each client. The wardens are subordinate to a type-independent component called the *viceroy*, which is responsible for centralized resource management.

The collaborative relationship in the application-aware adaptation is thus realized in two parts. The first, between the viceroy and its wardens, is data-centric: it defines the consistency levels for each data type and factors them into resource management. The second, between applications and Odyssey, is action-centric: it provides applications with control over the selection of consistency levels supported by the wardens.

The BARWAN Project In the BARWAN project, the application specific proxy uses the proxy agents to optimize the quality of service for the client in real time. To use transcoding to adapt to network variation, the proxy must have an estimate of the current network conditions along the path from the proxy to the client. SPAND (Shared Passive Network Performance Discovery), a network measurement system, allows a measurement host to collect the actual application-to-application network performance (e.g., available bandwidth and latency) between proxies and clients. SPAND monitors end-to-end bandwidth and connectivity to the clients (and servers) and notifies the proxy of any changes, which may result in changes in transcoding to adjust the quality of service. The original servers are unaware of the transformations or of the limited capabilities of the clients or networks.

The Prayer System In the Prayer System the application-aware adaptation is supported with the use of abstractions: QoS classes and adaptation blocks. A QoS class is defined by specifying the upper and lower bounds for resources. An application divides its execution into adaptation blocks. An adaptation block consists of a set of alternative sequences of execution, each associated with a QoS class. At the beginning of an adaptation block, an application specifies the QoS classes that it is prepared to handle, along with a segment of code associated with each class and an action to take should the QoS class be violated within the code segment.

4.2.3 Enhancing both Sides: The Mowgli Project

The Mowgli [76] communication architecture [60, 61] replaces the traditional client-server paradigm by a new client-mediator-server paradigm. As Figure 4.5 depicts, a mobile workstation connects to the fixed network through a wireless link. The Mobile-Connection Host (MCH) provides a service access point to Internet services.

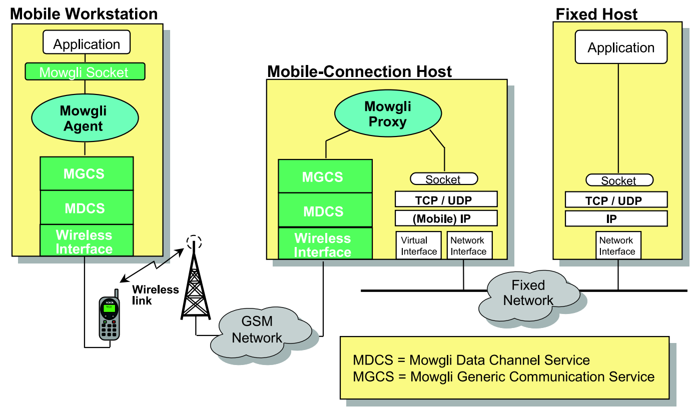


Figure 4.5: Overview of Mowgli Communication Architecture

Applications on a mobile workstation obtain the basic communication services through an application programming interface called the Mowgli socket interface. Since the Mowgli sockets are downward compatible with the Berkeley (BSD) sockets, existing applications using TCP or UDP sockets need neither be modified nor recompiled. The Mowgli socket interface binds the applications to the communication services in the Mowgli agent-proxy layer. A proxy, which runs in the MCH, and an agent, which runs in the mobile workstation, co-operate in the role of a mediator. An application on the mobile workstation sees the agent as its peer while an application in the Internet sees the proxy as its peer. Instead of TCP/IP the communication between the agent and proxy is based on the Mowgli Data Channel Service (MDCS) and on the Mowgli Generic Communication Services (MGCS).

The concept of the mediator is a convenient tool when problems due to wireless communication are solved in order to meet the needs of nomadic users. In the Mowgli approach an agent-proxy team is a distributed mediator. There are two basic kinds of agent-proxy teams: generic ones and application-specific ones. A generic agent-proxy team mediates application protocol data as it is and is able to support any existing application. A generic team can also provide generic enhancements, for example data compression, available in the MGCS. An application specific agent-proxy team is tailored for a certain application, like WWW. Such a team can exploit application semantics in optimizing the communication over the wireless link and take care of mobility-related functionality in the application.

The Mowgli Data Channel Service provides a flexible set of communication services to agents and proxies. The basic service includes bidirectional data channels that transfer data over the wireless link. In Mowgli there are two basic kinds of channels: stream channels, which provide TCP-like functionality, and message channels, which provide UDP-like functionality. In contrast to UDP Mowgli message channels provide a reliable datagram service.

Each channel has independent flow control and its own attributes that control the behavior of the channel. Data channels are multiplexed according to channel priorities. Even if there are bandwidth-intensive background transfers, priority-based scheduling allows the MDCS to provide reasonable response times to interactive applications. The MDCS also provides improved fault-tolerance. The MDCS is designed to recover efficiently from unexpected temporary disconnections which are quite common in cellular environments. The MDCS maintains the channel state so that the transfer can be resumed after interrupts. Each data channel has its own attributes that control the behavior of the channel.

4.2.3.1 Mowgli WWW

The Mowgli WWW software [65] consists of the following basic components shown in Figure 4.6: Mowgli WWW Agent, Mowgli WWW Proxy, Control Tool, and Apache HTTP server. One of the main principles in the Mowgli architecture is that there should be no need to modify existing applications. Mowgli WWW has been designed in accordance with this principle.

The Mowgli WWW Agent and Proxy cooperate to fetch hypermedia documents from fixed WWW servers to the mobile workstation. With each other they communicate using the highly optimized Mowgli HTTP (MHTTP) protocol. No modifications to WWW clients or servers are required. Together, the agent and proxy optimize the data traffic over the wireless part of the network for maximum performance.

The Control Tool is a separate program that is launched in conjunction with a conventional WWW browser. It provides a user interface to control

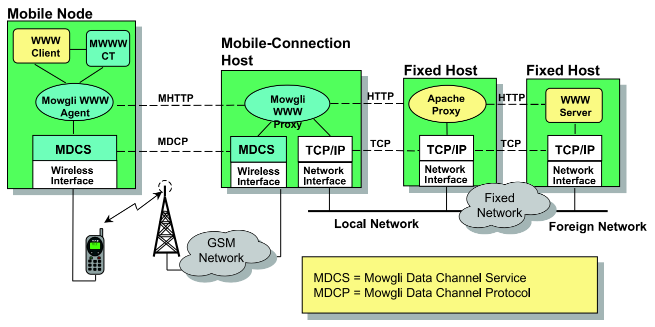


Figure 4.6: Overview of Mowgli WWW

the overall operation of Mowgli WWW. In addition, a per-document user interface is provided. This interface contains settings and operations together with status information that applies to individual documents. The primary advantages of this approach are:

Eliminating Extra Round Trips Removing various extraneous round trips inherent in the HTTP protocol is fairly straightforward with the mediators. The Mowgli WWW Agent intercepts requests from the local WWW client, translates them into the optimized MHTTP requests, and forwards them over the wireless link to the proxy. Posing as a client, the proxy, in turn, connects to the HTTP Proxy for each request and asks it to perform the request. The proxy then forwards the response back to the agent, which passes it on to the client. The Mowgli WWW Proxy makes extensive use of a predictive upload facility by prefetching document objects embedded in WWW documents. This means that the proxy knows that embedded document objects are going to be needed soon, and thus it uploads the objects to the mobile workstation without waiting for a specific request from the agent. This technique significantly reduces the response times observed by the end-user when fetching typical WWW documents.

Reducing Amount of Transferred Data The Mowgli WWW Proxy compresses document objects while sending them over the wireless link, and the Mowgli WWW Agent decompresses them before forwarding them to the WWW browser. In order to achieve efficient compression of transferred data Mowgli WWW supports content-type specific data compression: each particular document type (e.g., text, image data, audio data) can be assigned a different compression algorithm that performs best on that type of data.

Background Operations The background transfer feature in Mowgli WWW offers a convenient way for accessing WWW pages without the need to wait actively for the response. The Mowgli WWW Agent places the requests for background transfers into the batch transfer queue from which it proceeds to fetch them one by one. In order not to interfere with normal operation, data channels with a low priority are used. As the scheduling control is pre-emptive, the highest-priority channel with data to send always gets link capacity. This ensures that the user always gets prompt response, even when there are on-going background transfers.

The user can look at the jobs in the queue, start and stop the queue at will, delete an individual job or move one in front of the queue. The background fetches can also be started at a preset time, for instance at a time when the telephone calls are cheaper.

4.3 Addressing the Mobility Issues

4.3.1 Mobile IP

When IP routing was originally defined, mobility of hosts was not considered to be an issue. Routing methods were built for static networks, where the hosts were unlikely to move from one subnet to another. Routing takes advantage of a *network number* contained in every IP address. Thus, the IP address encodes the computer's physical location, and, by default, the location is fixed.

Mobile IP [83] defines protocols and procedures by which packets can be routed to a mobile node, regardless of its current point-of-attachment to the Internet, and without changing its IP address. Mobile IP consists of the following entities:

Mobile Node (MN) A host or router that may change its point of attachment from one network or subnetwork to another through the Internet. A mobile node must support mobile IP in order to communicate with mobile agents, the home agent and the foreign agent.

Home Agent (HA) A home agent is always attached to the home network of a mobile node, that is, the network that a mobile node belongs to by its IP address. A home agent must support mobile IP and is responsible for forwarding packets destined to the mobile node at the network that the mobile node is attached to. Forwarding is accomplished by IP-in-IP tunneling.

Foreign Agent (FA) Situated in the foreign network, it handles registration requests and replies between the mobile node and the home agent, and is usually the other endpoint of the IP-in-IP tunnel.

Care-of-address (COA) An address which identifies the mobile node's current location. It can be viewed as the end of a tunnel directed towards a mobile node. It can be either assigned dynamically or associated with its foreign agent.

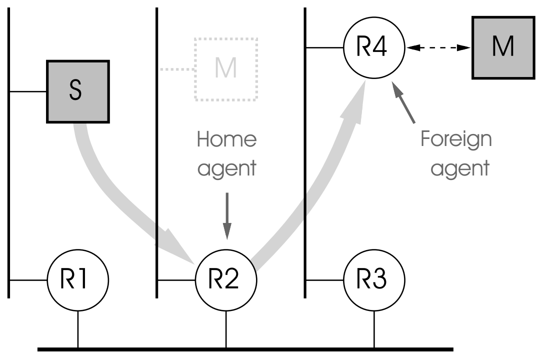


Figure 4.7: Mobile IP entities

There are several actions to be taken to allow a Mobile node to move.

4.3.1.1 Agent Discovery

This protocol, based on ICMP, provides a means for a mobile host to detect when it has moved from one network to another, and for it to detect when it has returned home. When moving into a new foreign network, the agent discovery protocol also provides a means for a mobile host to discover a suitable foreign agent in this new network with which to register.

Home agents and foreign agents periodically advertise their presence by multicasting an agent advertisement message on each network to which they are connected (Figure 4.8). Mobile hosts listen for agent advertisement messages to determine which home agents or foreign agents are on the network to which they are currently connected. If a mobile host receives an advertisement from its own home agent, it deduces that it has returned home and registers directly with its home agent. Otherwise, the mobile host chooses whether to retain its current registration or to register with a new foreign agent that it knows.

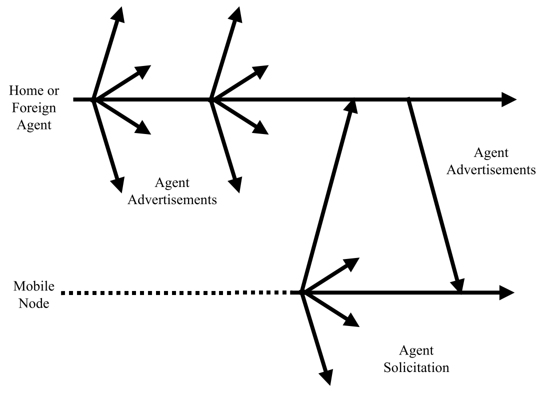


Figure 4.8: Agent Discovery Protocol

While at home or registered with a foreign agent, a mobile host expects to continue to receive periodic advertisements from its home agent or from its current foreign agent. If that does not happen, the mobile host may deduce either that it has moved or that its home agent or current foreign agent has failed. If the mobile host has recently received other advertisements, it may attempt registration with one of those foreign agents. Otherwise, the mobile host may multicast an agent solicitation message onto its current network.

4.3.1.2 Registration

When connected with a new foreign agent, a mobile host must register with that foreign agent. It must also register with its home agent to inform it of its new care-of address. Furthermore, when a mobile host returns to its home network, it must register with its home agent to inform it that it is no longer using a care-of address.

There are two possible scenarios. In the first case the mobile node receives an agent advertisement and discovers a care-of address from the agent advertisement extension. The mobile node sends a registration request to the foreign agent including the mobile node's home address, the home agent's address, the mobile node's care-of address (Fig. 4.9). The foreign agent receives the registration request and after inspecting it either relays the message to the home agent or replies to the mobile node with a denying registration reply. A possible reason for a denying reply might be too big a registration lifetime value in the registration request.

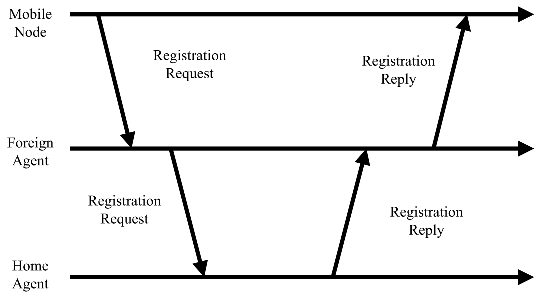


Figure 4.9: Registering through a Foreign Agent

In the second case the mobile node discovers that its own IP address has changed. This change might have happened automatically or by user intervention. The new IP address is called a co-located care-of address. The mobile node constructs a registration request that contains the same information as in the first case, except that the care-of address is replaced with the co-located care-of address. The mobile node sends this registration request straight to the home agent. Thus a mobile node can also roam a network that does not offer foreign agent services.

4.3.1.3 In Service

After the registration is completed a IP-in-IP tunnel is set up between the home agent and the care-of address or the co-located address.

The Mobile IP requires support for *IP-in-IP encapsulation* for tunneling, as illustrated in Figure 4.10. In this method, to tunnel an IP packet, a new IP header is wrapped around the existing packet; the source address in the new IP header is set to the address of the home agent, and the destination address is set to the mobile host's care-of address. The new header added to the packet is shaded in gray in 4.10. This type of encapsulation may be used for tunneling any packet, but the overhead for this method is the addition of an entire new IP header (20 bytes) to the packet.

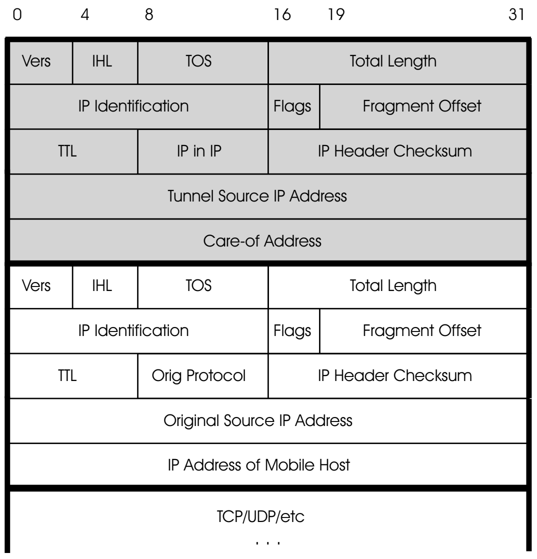


Figure 4.10: IP in IP encapsulation

Mobile IP also defines a more efficient encapsulation, called *minimal encapsulation*. Minimal encapsulation is an encapsulation method that is very efficient in terms of overhead: it adds only 8 or 12 bytes to each packet sent to a mobile host. In the encapsulation process the header is rewritten. The outer packet has a standard IP header, while the encapsulated packet has a dedicated but small header. This is why it is said that the tunneling header is inserted immediately after the original header of Figure 4.11.

4.3.1.4 Deregistration

After the mobile node returns home, it deregisters with its home agent to drop its registered care-of address. There is no need to deregister with the foreign agent because the service expires automatically when the service time expires.

4.3.2 Mobility Support in IPv6

The design of Mobile IP support in IPv6 [27, 56] has been highly influenced by the IP support in IPv4 (Mobile IP). Mobile IPv6 thus shares many features with Mobile IP, but the protocol is now fully integrated into IP and

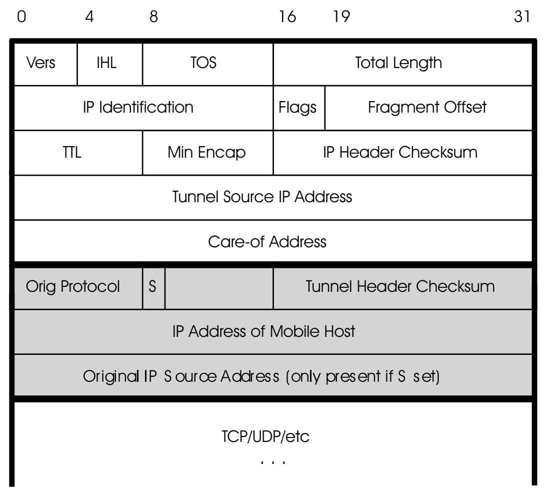


Figure 4.11: Minimal encapsulation

provides many improvements. This section summarizes the main differences between Mobile IP and Mobile IPv6.

- IPv6 supports the *Route Optimization* functionality as a fundamental part of the protocol, rather than being added on as an optional set of extensions as in Mobile IP. This integration of Route Optimization functionality allows direct routing from any correspondent node to any mobile node, without needing to pass through the mobile node's home network and be forwarded by its home agent. The Mobile IP *registration* functionality and *Route Optimization* functionality are performed by a single protocol rather than two separate (and different) protocols.

- In Mobile IPv6 a mobile node uses its care-of address as the Source Address in the IP header of packets it sends, allowing the packets to pass normally through ingress filtering routers. The home address of the mobile node is carried in the packet in a Home Address destination option, allowing the use of the care-of address in the packet to be transparent above the IP layer.

- There is no longer any need to deploy special routers as *foreign agents*. In Mobile IPv6, mobile nodes make use of IPv6 features, such as *Neighbor Discovery* and *Address Autoconfiguration*, to operate in any location away from home without any special support required from its local router.

- Most packets sent to a mobile node while away from home in Mobile IPv6 are sent using an IPv6 Routing header rather than IP encapsulation, whereas Mobile IP must use encapsulation for all packets. This requires a smaller header reducing the overhead.

- While a mobile node is away from home, its home agent intercepts any packets for the mobile node that arrive at the home network, using IPv6 *Neighbor Discovery* rather than ARP. The use of Neighbor Discovery improves the robustness of the protocol and simplifies its implementation.

- The dynamic home agent address discovery mechanism in Mobile IPv6 uses IPv6 *anycast* and returns a single reply to the mobile node, rather than the corresponding Mobile IP mechanism that used IPv4 directed broadcast and returned a separate reply from each home agent on the mobile node's home link. This mechanism is more efficient and more reliable, since only one packet need be sent back to the mobile node.

- Mobile IPv6 defines an *Advertisement Interval option* on Router Advertisements allowing a mobile node to decide for itself how many Router Advertisements (Agent Advertisements) it is willing to miss before declaring its current router unreachable.

4.4 Enhancing the Middleware Layer

4.4.1 Dolmen

Wireless access and terminal mobility issues in CORBA have been studied in the EC/ACTS projects DOLMEN [66].

Dolmen uses the concept of bridging to interconnect mobile terminals to the fixed network. In particular, it implements two half-bridges, one residing in a mobile terminal and the other in a well-known access point within each mobility domain in the fixed network. This allows the introduction of an efficient light-weight Inter-ORB protocol for use over the wireless access network. This approach also allows addressing terminal mobility, performance, and reliability issues.

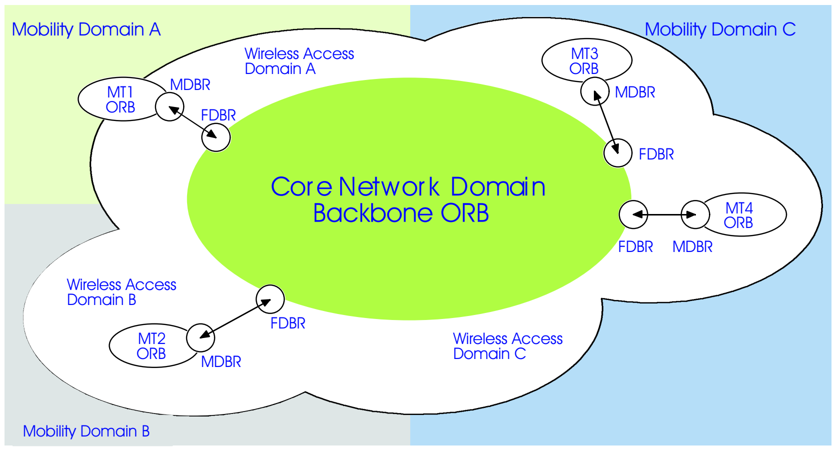


Figure 4.12: Dolmen Architecture

Figure 4.12 shows how mobile terminals can be connected to a fixed network domain with mediated bridging: The wireless access domain and part of the core network domain is divided into mobility domains. The core network part of each mobility domain instantiates one or more Fixed DPE Bridges (FDBRs) that serve as access points to the fixed network. Each mobile terminal has its own ORB that provides object services to the applications running on the terminal. Invocations of objects outside the local access domain are directed to the Mobile DPE Bridge (MDBR) on the mobile terminal.

The MDBR forwards the invocation to the FDBR, which then invokes the desired object. The FDBR acts as the representative of the mobile terminal within the fixed network, invoking operations in other objects on behalf of the mobile terminal. The FDBR also accepts invocation requests for objects located on the mobile terminal from objects within the core network. The FDBR forwards an invocation request to the MDBR, which then invokes the actual object and returns the response through the FDBR.

4.4.1.1 Allowing Terminal Mobility

When a mobile terminal is in contact with the core network (Figure 4.12), a physical signaling connection exists between the two bridges. When the terminal moves to another domain, this signaling connection is released, a new FDBR within the new domain is contacted, and a new signaling connection created. In Dolmen this procedure is referred to as a *bridge handover*. During a bridge handover, the Location Register is updated and the new FDBR is registered as the current access point of the mobile terminal. This allows future invocation requests to be routed to the mobile terminal through the correct FDBR.

Since one of the basic requirements for the mobility bridges is that they must hide the effects of mobility from client and server objects, the invocations in progress at the time of handover must be reliably and correctly completed despite the momentary break in connectivity that is inherent in a handover. This reliability is achieved by buffering invocation-related messages in the old FDBR until the mobile terminal has successfully connected to a new FDBR. A forwarding connection is then set up between the old and new FDBRs and the buffered messages are forwarded to the new FDBR.

Because IIOP is a connection-oriented protocol and an invocation reply is always sent over the same connection from which the request arrived, it is not possible to re-route an invocation reply. Therefore, the tunnel connection is maintained until replies for all pending invocations have been delivered to their destination through the old FDBR. Since CORBA communication requires a reliable message service, the bridges must perform recovery after an unexpected loss and subsequent re-establishment of the signaling connection. The LW-IOP protocol described below provides the means for such a recovery: each LW-IOP message must be acknowledged by the receiver before it can be discarded by the sender. In the event of a communication error, any unacknowledged messages are resent after the communication channel has been re-established.

4.4.1.2 Light-Weight Inter-ORB Protocol (LW-IOP)

Since all object invocations between a mobile terminal and the core network pass through MDBR and FDBR, as depicted in Figure 4.13, it is possible to optimize the messages passing through the wireless connection. Dolmen designed a special Light-Weight Inter-ORB Protocol (LW-IOP) for wireless access networks with low-bandwidth signaling channels.

The LW-IOP protocol is based on the GIOP protocol in the sense that the exact same functionality is supported. However, the set of messages and

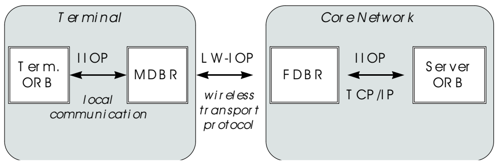


Figure 4.13: Protocols in invocations over a wireless network

their representations are more efficient. In addition, caching and compression techniques are utilized in order to reduce the number of octets sent over the wireless access network.

The main properties of LW-IOP are:

- Reduced size of structures. For example, GIOP request identities are allocated 32 bits, thus allowing for the unlikely scenario of more than 4 billion concurrent requests. LW-IOP request identities permit 16,384 concurrent requests, and thus require only 14 bits.
- Reduced size of headers. For example, the basic type `unsigned long` is frequently used, for instance to indicate the length of a string. Since these values are rather small in general, a representation with a variable size is used, in 1 to 5 octets instead of systematically 4 bytes. In most cases, the numbers are represented in 1 or 2 octets.
- Strip of protocol identifiers and version numbers from the headers.
- Coded representations of textual and binary strings, which are very usual in GIOP, are employed to avoid the need to always transmit the full information.

4.4.2 Wireless CORBA

OMG itself recognized the problems of CORBA in wireless and mobile domains and thus issued a Request For Proposal [81] to solicit proposals that allow applications on mobile terminals to exploit and provide CORBA-based services. The main issues to resolve were:

1. IIOP servers are not expected to change their transport connection endpoints.

2. Both IIOP and the transport connections are supposed to be reliable. IIOP has no support for resuming broken IIOP connections, and if the connection breaks the client and server are left in an inconsistent state.

3. There are no means of changing network interface during an IIOP connection without breaking it.

4. As pointed out in the previous section, the connections are expected to enjoy high bandwidth.

Consequently to the RFP the OMG has adopted the *Wireless Access and Terminal Mobility in CORBA* [32] specification, highly influenced by the Dolmen project.

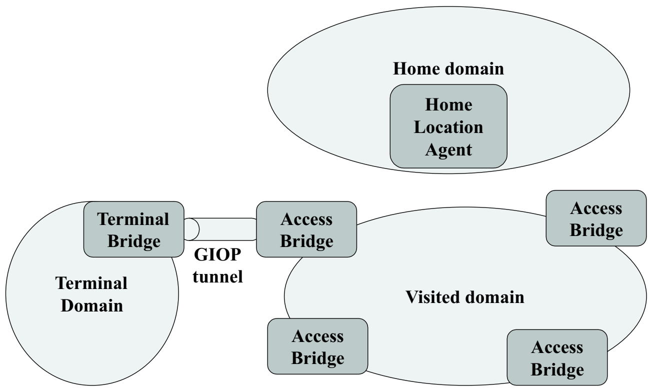


Figure 4.14: Architecture for Terminal Mobility in CORBA

4.4.2.1 Architectural Framework

The key elements in the architecture are shown in Figure 4.14. It identifies three different domains: the Home Domain, which hosts the Home Location Agent, the Visited Domain, which hosts one or more Access Bridges and the Terminal Domain, which hosts the Terminal Domain. These different concepts are described below.

Mobile IOR A mobile IOR is a special IOR that hides the mobility of a terminal from clients that invoke operations on objects located on the terminal. The Mobile IOR provides this feature is a transparent way to the client's ORB. Hence a non-mobile client does not need to implement the Wireless CORBA specifications.

Home Location Agent The role of the Home Location Agent is to keep track of the current location of the terminal. It provides operations to query and update terminal locations.

Access Bridge The Access Bridge is the network end point of the GIOP tunnel. It encapsulates the messages sent to the Terminal Bridge and decapsulates the messages coming from the Terminal Bridge. If needed the Access Bridge may also provide notifications of terminal mobility events.

Terminal Bridge On the terminal side of the GIOP tunnel there is the Terminal Bridge. It encapsulates the messages sent to the Access Bridge and decapsulates the messages received from the Access Bridge. The Terminal Bridge may provide a channel that delivers terminal mobility events.

GIOP tunnel The GIOP tunnel is the mean to transmit messages between the Terminal Bridge and the Access Bridge. The GIOP Tunneling Protocol (GTP) defines how GIOP messages are transmitted and the control messages to establish, release and re-establish the GIOP tunnel. The GTP is a transport-independent protocol. The specification defines three concrete tunneling protocols over TCP, UDP and WDP. The overall architecture is depicted in the figure 4.15.

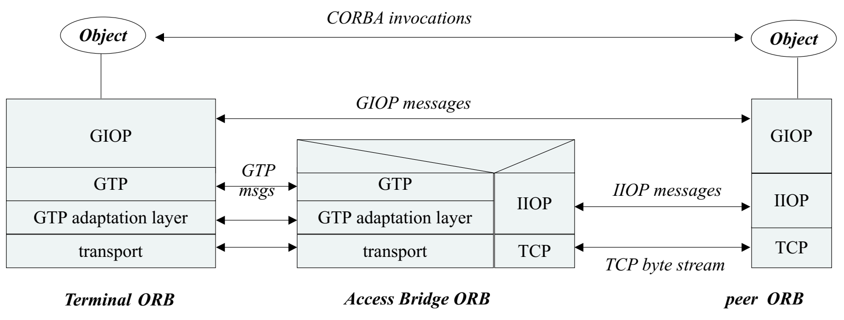


Figure 4.15: GIOP Tunneling Protocol Architecture

4.4.2.2 Message Processing

When the Home Location Agent receives a message targeted at a terminal, it returns the Mobile IOR of the Address Bridge associated with that terminal. If the Home Location Agent does not know the Address Bridge it returns a system exception.

When the Access Bridge receives a message targeted to the terminal it is associated with, it encapsulates the message with the GTP and sends it through the GIOP tunnel.

However, if the Access Bridge receives a message aimed to a terminal it has not been associated with, then it queries the current location of the terminal from the Home Location Agent.

4.4.2.3 Handoff

The handoff in Mobile CORBA can occur in two different cases: backward handoff (Bridge handoff) and forward handoff (access recovery). It may have different initiators: Network Initiated handoff, when an external application invokes the start of the handoff, or Terminal Initiated handoff, when the terminal connects to a new Access Bridge.

In general, a stretch of the handoff sequence is the following (Fig. 4.16):

1. The old Access Bridge requests the new Access Bridge to accept the terminal.
2. The old Access Bridge communicates to the Terminal Bridge to establish a connection to the new Access Bridge.
3. The new Access Bridge updates the location of the terminal to the Home Location Agent.
4. The Terminal Bridge communicates to the old Access Bridge that a connection has been established.
5. The old Access Bridge communicates to the new Access Bridge that the handoff is completed and releases the GIOP tunnel with the Terminal Bridge.

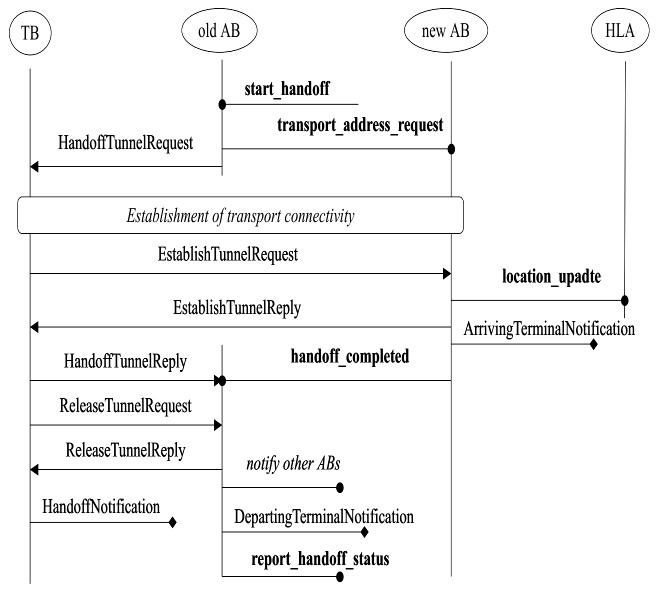


Figure 4.16: Sequence of Handoff

4.4.2.4 Access Recovery

When the Terminal Bridge detects that the connectivity with the Access Bridge is lost it starts the access recovery procedure. The possible outcomes of the procedure depend upon the fact that a connectivity is established with the same Access Bridge or with a new Access Bridge. In the first case, after the connection is secured a retransmission of the pending messages takes place. In the latter case, the retransmission will take place through the new Access Bridge after an implicit handoff.

4.4.3 Alice Project

The Alice project [3, 46] tries to address the same problems identified in the previous section. The ALICE architecture allows server and client objects to reside on mobile terminals without relying on a centralized location register to keep track of their movements. In ALICE IIOP clients and servers on the mobile terminal can interact with IIOP servers and clients on the wired network using standard IPv4 transparently. In a similar way as the Access Bridge of the previous section, a *Mobility Gateway* is situated on the wired network to allow the mobile terminal to access the wired network. The architecture is depicted in Figure 4.17. The architecture consists of three layers. The *Mobility Layer* (ML) provides mobility support independently from CORBA and IIOP. The *IIOP Layer* implements the IIOP protocol independently from the mobility. The *Swizzling IIOP* (S/IIOP) provides to IIOP the support required when the server objects reside on mobile terminals.

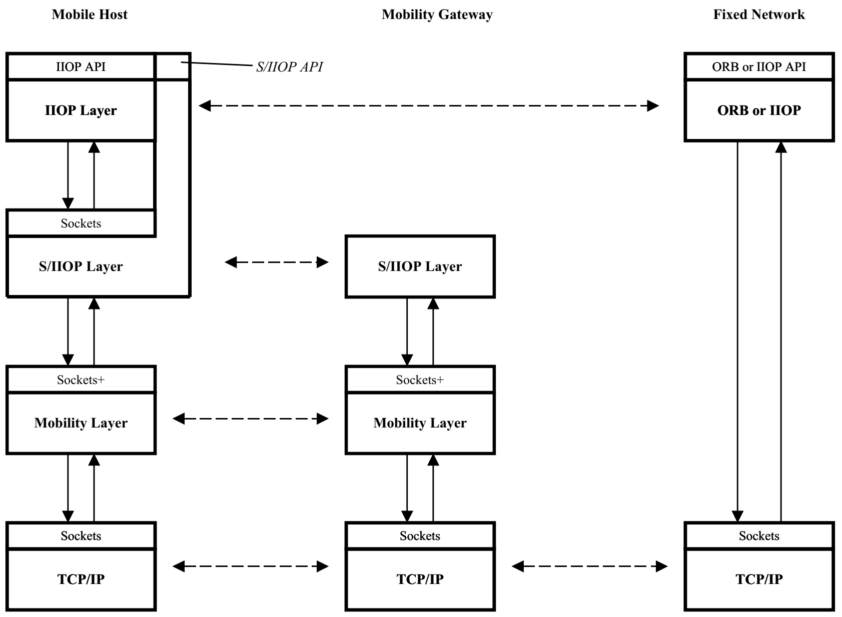


Figure 4.17: The ALICE Architecture

4.4.3.1 Mobile Layer

The Mobile Layer hides the broken connection from the layers above it, thus avoiding breaking the IIOP semantic. Furthermore, the ML allows the mobile terminal to allocate TCP/IP ports on the Mobility Gateway. This is needed for incoming connection attempts. The ML offers mobility information to the S/IIOP layer on both the mobile terminal and the mobility gateway, so that address translation and request forwarding can be performed. Finally the ML performs a handoff between different gateways.

4.4.3.2 Handoff

When the Mobile Layer on the mobile terminal connects to a new ML on the Mobility Gateway (MG) a handoff takes place. Since in the ALICE framework there is not a centralized registry, the new MG is informed by the mobile terminal of the address of the old MG and the identifiers of all the logical connections that existed between the mobile terminal and the old MG. The old MG sends all the unacknowledged data it has in its cache to the new MG. If there were open connections between the old MG and the mobile terminal, they must *all* be tunneled between the old and the new MG. This could lead to a chain of tunnels between different mobility gateways if the terminal moves frequently. The Swizzling Layer performs all the needed update operations to the IORs.

4.4.3.3 Drawbacks

The idea not to rely on a centralized location service is interesting but there are two main drawbacks in the ALICE architecture. First, its implementation required a modification to the standard socket semantics. This is because it is not possible for a server located in a mobile terminal to bind to a particular port number. Secondly, the *chain of tunnels* that may occur when the mobile terminal changes its point of access frequently does not sound appealing.

Chapter 5

Wireless Java RMI

Complex problems have simple, easy-to-understand wrong answers.
- Grossman's Misquote Of HL.Mencken

5.1 Introduction

Java RMI (see Par. 3.3) is becoming extremely popular in distributed computing when the language environment is Java. The reasons behind this success are quite obvious: RMI alleviates the user from the burden of all the communication between remote objects. With only few modifications to the source code the application is easily transformed to a distributed one. RMI itself takes care of the details as memory management and distributed naming. Furthermore, given that Java is becoming the de-facto standard language over the Internet, developers are encouraged to reuse off-the-shelf solutions as RMI.

This popularity is also reaching new environments: RMI is a popular choice in Agent Platform for extra-platform communication, and it is the default communication tool in the increasingly popular middleware Jini (see 6.3.3). Thus Java RMI is used more often in mobile environments, such as in mobile agent platforms, and in wireless environments, such as some Jini scenarios suggest. Thus, it is only a matter of time before the performance of Java RMI over wireless links becomes important.

For this reason, and to show in more detail how much the nomadic environment affects the behavior of the non nomadic-aware software, we analyzed the performance of Java RMI over a slow wireless link.

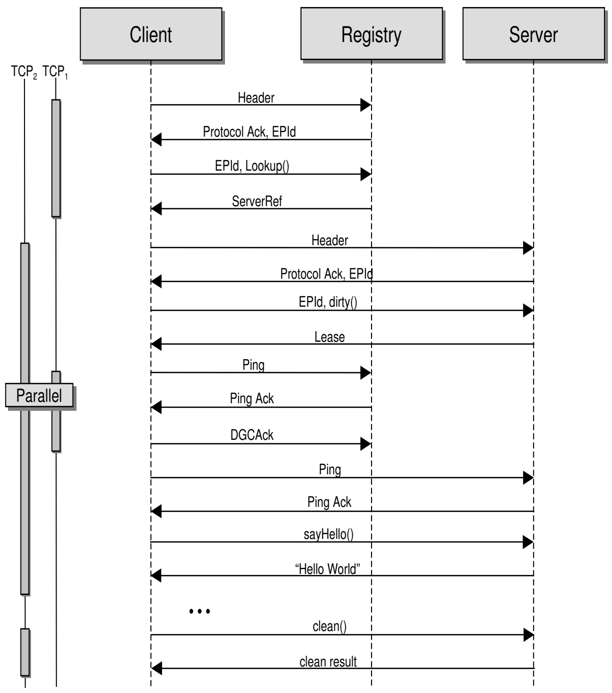


Figure 5.1: The trace of the "sayHello" remote invocation

5.2 RMI Problems

In this section we describe the main problems that RMI shows in a wireless environment. The data was collected using a network sniffer and then analyzed.

5.2.1 Analysis of an RMI Call

In this section we will show an analysis of a simple RMI call. The method is called "sayHello". As return value the method returns the String "*Hello World*".

In our test the stub class was already present on the client side, so there was no need to download it. The trace of the call is outlined below (see Figure 5.1):

1. The first round-trip is between the client and the Registry on the remote side and uses a new TCP connection(*TCP1*). The registry returns an acknowledgment of the RMI protocol and what it believes

to be the client's IP address (EPId). It should be noted that this first round-trip happens every time a new connection is opened.

2. In the second round-trip, the client requests (and obtains) the remote reference of the desired remote class. At this point, the RMI connection is closed, logically closing the TCP1 connection.

3. Opening a second TCP connection *TCP2*, the client connects with the server. Since this is the first RMI call a header and a protocol acknowledgment are exchanged.

4. The client-side Distributed Garbage Collection (DGC) requests from the server a lease of the required remote reference through a *Dirty()* invocation. The RMI connection is closed, logically closing the TCP2 connection.

5. At this point, the Client must tell the DGC of the Registry that it obtained a remote reference, so it opens a new RMI connection. The first round-trip between the client and the Registry is a ping: in this way the client verifies that the TCP connection, which was logically closed before, is still alive[1]. Having verified this, the client communicates to the Registry that it has received a lease from the server with a DGCAck message.[2]

6. In parallel with the previous point, the client can invoke the remote method on the server. But, since the TCP connection was closed, a ping round-trip takes place. After this, the client invokes the method and obtains the results of the invocation as return value.

7. When the client does not need the remote reference any more, usually when the remote reference is locally unreferenced, it sends a "clean" message to the server. This exchange is preceded by the usual ping round-trip.

Data traffic is summarized in Table 5.1. In each row is given the number og bytes of data (and percentages) transferred in each transfer pattern.

[1]Note that a ping does not occur if the TCP connection has been idle for a time less than a ping round-trip.

[2]Before this acknowledgment has been sent, the Registry must not release its server reference, since that might cause the server to wrongly conclude that there are no references in use.

Table 5.1: Invocation data traffic

	Client to Server and Registry	Server and Registry to Client	Total
Registry Lookup	55 (6%)	276 (42%)	331 (20%)
Invocation Data	41 (4%)	37 (6%)	78 (5%)
DGC Data	831 (85%)	305 (46%)	1136 (69%)
Protocol Overhead	52 (5%)	40 (6%)	92 (6%)
Total	979 (100%)	658 (100%)	1637 (100%)

On a slow wireless link the amount of data that is sent over the link is important. In this example, the actual invocation takes up only 5% of the total transmitted data while 69% was related to the DGC protocol. This means that the channel is primarily used for auxiliary data, making the invocation expensive.

Another important issue is the high number of round-trips. On slow links, like GSM, even a single byte exchange like ping causes delays due to the long latency times involved (a round-trip over GSM is typically around one second). In this example, six round-trips were necessary before the invocation was completed, not counting the two round-trips caused by TCP handshaking. However, only two are really needed - one to get the server reference, and another for the actual invocation.

5.2.2 RMI Use of TCP Connections

The reuse of TCP connections in the transport layer is commendable because it saves resources. However, the implementation causes frequent ping messages. This is problematic with high-round-trip wireless connections. While Java RMI is not optimal for wireless networks, neither is TCP upon which Java RMI is built. The problems with TCP in a wireless environment are well-known [61, 18]. Especially in JDK1.1, RMI and TCP conspire to produce bad results. RMI writes header data byte by byte, and because of the slow start algorithm [54], TCP has to wait for an acknowledgment once it has sent the first segment containing only one byte. This means that it takes a full round-trip before the second segment can be sent.

In JDK1.2 (Java2), data is no longer written byte by byte, and the performance is much better. However, the protocol itself still enforces many round-trips for a single invocation (due mainly to the ping packets).

5.3 Optimization of Java RMI for Slow Wireless links

Tests run on top of a GSM data connection show that a simple reference lookup can take as long as 20 seconds for JDK1.1. This fact should not come as a surprise to the reader at this point as we have seen in the previous chapters. Given the importance of Java RMI, we decided to optimize it for wireless links. The following sections describe the solutions we designed and outline the reasons behind our choices.

5.3.1 Optimizations

Optimization of Java RMI for wireless links means a reduction of protocol overhead and the number of round-trips. The analysis of the trace of the RMI call shown in Figure 5.1 suggests the following optimizations:

Serialization – The serialization protocol used by Java RMI produces a large amount of overhead in the invocation. This overhead should be minimized.

Protocol Acknowledgment – It should be possible to suppress protocol acknowledgments, or at least to avoid their crossing the wireless link.

Registry lookups – Lookup invocations are very expensive in terms of data transferred through the wireless link. Thus lookups should be avoided as much as possible.

Distributed Garbage Collection – This protocol introduces heavy data overhead and its use in a wireless environment is less meaningful, since the link is subject to sudden disconnections that can be handled at the transport layer. It is important to avoid its redundancy and high number of round-trips.

5.3.2 Maintaining Compatibility

Once identified where to focus on to improve RMI performance, there are two different ways to implement the solutions. The first one is to change the RMI implementation itself, i.e. to change JDK system classes. The main disadvantage of this choice is that modifications of both client and server software is necessary, thus making this solution unattractive. In fact, while it can be acceptable to modify the runtime classes on the client side, it is inconceivable to modify the runtime classes on the server side. In fact, the client's host is in possession of the user, while the servers' hosts can be spread through the Internet and thus outside the direct influence of the user.

The second solution, presented in the next sections, attempted to preserve the original implementation supporting it with the use of mediators.

5.3.3 Use of Mediators

Figure 5.2 shows the scenario where a user invokes a remote method from a mobile device. There are three main actors in the scenario:

1. The user device, where the client invokes the RMI method, situated in a wireless domain.
2. The *access node*, situated in the fixed network, that serves the mobile terminal with an access point to the fixed network.
3. The server computer, where the server application is running, situated somewhere in the Internet.

It is possible to insert a mediator in the access node and a mediator in the user device. In this way we can decouple the RMI protocol from the wireless connection. On the client side, the RMI Agent captures the invocations made by the client, while in the access node the RMI Proxy forwards the requests to the server. In this way the two mediators can control the data passing through the wireless channel.

The role of the RMI mediators in an invocation is shown in Figure 5.3. The RMI Agent captures the invocation made by the client. A lookup request is first checked in the local cache, and only if the remote reference is unknown the request is forwarded to the server.

DGC invocations are optimized by decoupling client and server. The RMI proxy keeps servers alive by periodically renewing the leases[3]. It will only

[3]A lease is a time period after which the server will assume that the client has died unless the client renews the lease.

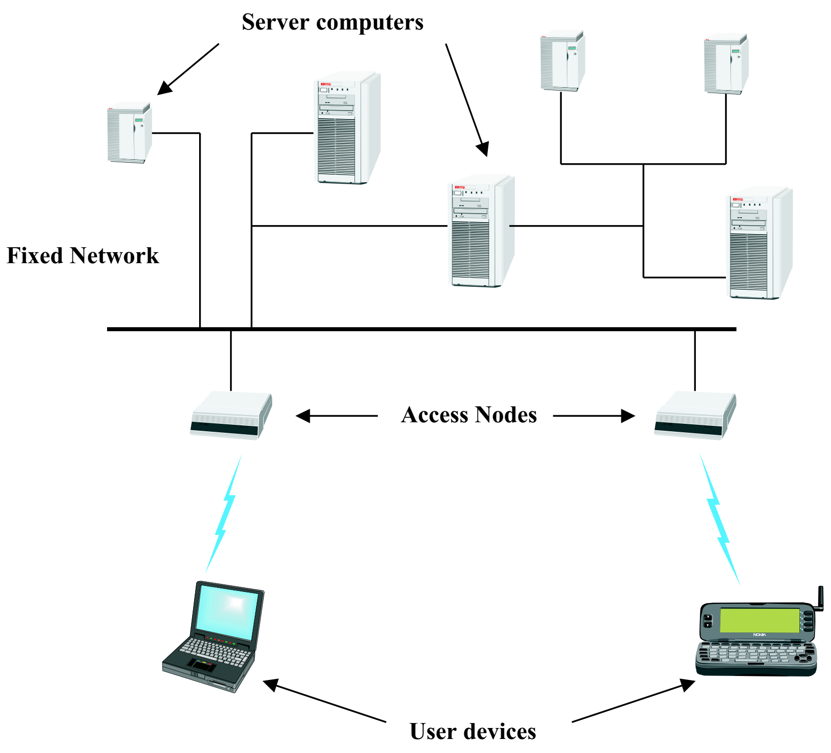


Figure 5.2: Wireless RMI scenario

stop doing so once the RMI agent tells it that no more references to the server exist on the other side. In this way the DGC semantics are loosened to suit the needs of wireless communication without modifying client or server code. No lease request, that is, *dirty()* invocations, needs to be sent over the wireless link since all leases are managed on the fixed network side. In this way the number of round-trips is minimized. The amount of data transferred over the wireless link is also reduced. Of course, clean requests still have to be sent to inform the other side that there is no reference to a server any longer. An optimized data representation can be used for these requests, which further reduces DGC overhead.

5.4 Implementation Details

Using mediators it is possible to maintain standard RMI compatibility between the client and the RMI Agent and between the RMI Proxy and the

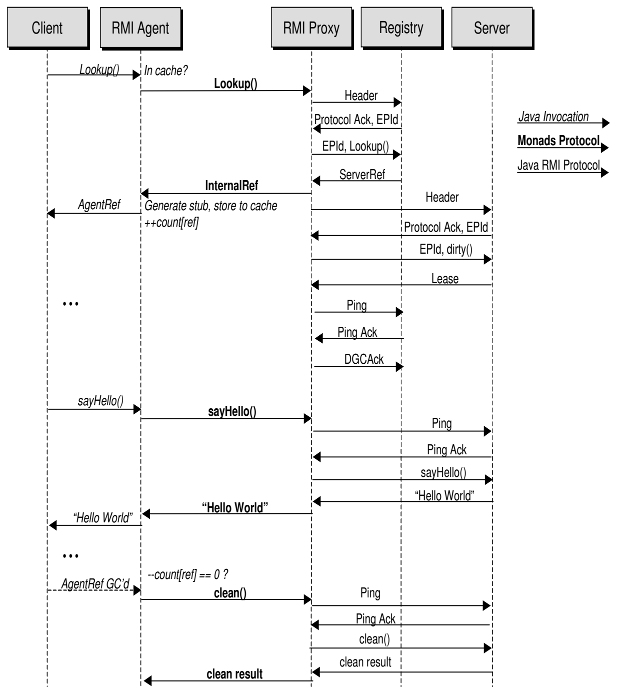


Figure 5.3: Using mediators to optimize the remote invocation

server. Practically this means that the RMI Agent is seen as a RMI server by the client, and the RMI Proxy acts as a client from the server's point of view. The main advantage of this approach is that the applications both in the user domain and in the server domain do not need to be modified. Unfortunately it is not an easy task to capture the invocations made by the client. As described in section 3.3, before invoking a remote method a client has to obtain the remote object reference. This is done usually by connecting to a local registry. This problem has been solved by modifying the runtime classes on the mobile device. Instead of a standard RMI registry the user application invokes our own version implemented inside the RMI Agent. It is the task of the Agent to request the reference from the Proxy.

A second problem arises when the reference is passed to the client as a stub. The stub has an open and direct connection with the remote server. Unfortunately following this path the client would bypass the RMI Agent and its optimization during the remote calls. The next section describes how we dealt with this issue.

5.4.1 Dynamic Run-time Generation of Generic Stub

To overcome the problem described in the previous session the RMI Agent has to return a generic stub to the client so that all the methods invoked by the stub are captured. The generic stub will contact the RMI Agent instead of connecting directly to the remote server. In order to make this possible, a dynamic run-time generation of classes must be possible. Fortunately JDK 1.3 introduced the concept of *dynamic proxy classes*.

A dynamic proxy class is a class that implements a list of interfaces specified at runtime when the class is created. A proxy instance is an instance of a proxy class. Each proxy instance has an associated invocation handler object, which implements the interface `InvocationHandler`. A method invocation on a proxy instance through one of its proxy interfaces will be dispatched to the invoke method of the instance's invocation handler, passing the proxy instance, a `java.lang.reflect.Method` object identifying the method that was invoked, and an array of type `Object` containing the arguments. The invocation handler processes the encoded method invocation as appropriate and the result that it returns will be returned as the result of the method invocation on the proxy instance.

To intercept the remote invocation our implementation follows these steps:

1. Obtain the list of the interfaces implemented by the remote object.
2. Attach a generic invocation handler to any method declaration in any interface implemented by the remote object.
3. Return a proxy instance to the client.

A segment of the invocation handler source code is shown in Program 5.1.

Each proxy instance has an associated invocation handler. When a method is invoked on a proxy instance, the method invocation is encoded and dispatched to the `invoke` method of its invocation handler.

The `DynStub` class implements the interface Invocation Handler so that the client's invocations in the stub are dispatched to its *invoke* method. The solution to our problems is in the last line of the invoke method, where

Program 5.1 DynStub.java

```
public  class DynStub
    implements InvocationHandler {

    private static int position;

    DynStub(int p){
        this.position=p;
    }

...

   public Object invoke(Object proxy, Method m, Object[]
         args) throws Throwable {

        if (args==null){
            args=new Object[0];
        }

        int methodindex = getMethodIndex(m,proxy);
        return HandlerClass.doInvoke(this.position,
              methodindex, args);
    }
}
```

the invocations are forwarded to the `doInvoke` method in the `HandlerClass` (segments of the code are shown in Program 5.2) implemented in the RMI Agent.

When a client requests a remote reference, the RMI Agent will invoke the `lookup` method in the `HandlerClass` class. This method obtains the list of the interfaces from the server side and creates a proxy using the `Proxy.newProxyInstance` method. When the client invokes a method, the proxy will invoke the method handler `doInvoke()` in the `HandlerClass`.

Note that, as side effect, no stub classes are downloaded in the client side through the wireless channel, since the real stub is returned by the remote server to the RMI Proxy in the server side and there it resides. The dynamic stub needs only the runtime classes of the interfaces implemented by the remote object, but those classes must be known by the client before making any remote method invocation.

Program 5.2 HandlerClass.java

```
public  static Remote lookup(String url, int port) throws
        NotBoundException,RemoteException {

    ...

    Class[] interfaces = new Class[n];

    ...

    interfaces= getRemoteInterfaces();

    InvocationHandler handler = new DynStub(position);
    Remote fakeStub = (Remote) Proxy.newProxyInstance(
           interfaces[].getClassLoader(),interfaces,handler
           );

    return fakeStub;

}

protected static Object doInvoke(int position,int
       methodindex,Object[] args) throws
       ClassNotFoundException,RemoteException {

    ...
}
```

Figures 5.4 and 5.5 outline the difference between the architecture of our RMI optimization and the normal RMI.

5.5 The Wireless RMI Protocol (ωRMI)

The Wireless RMI Protocol (ωRMI) is the core of the Wireless RMI architecture and its purpose is to provide an efficient means for communication between the RMI Agent and the RMI Proxy via the wireless channel. In the following section we describe in detail the messages that compound the protocol and how they are used.

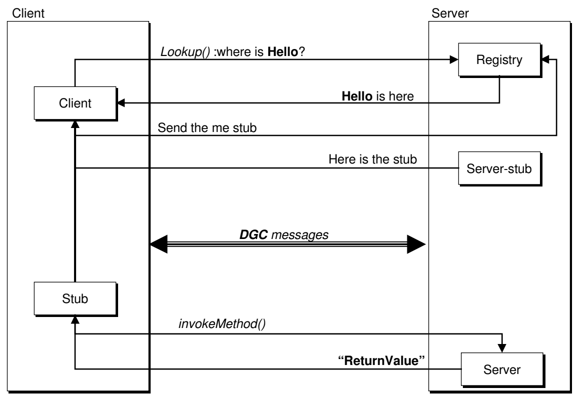


Figure 5.4: The normal RMI structure

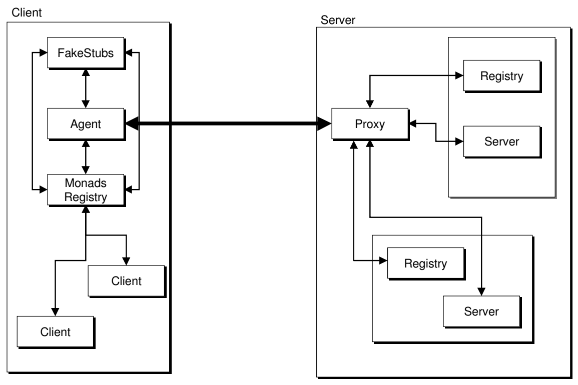


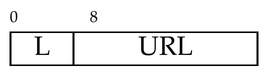
Figure 5.5: Optimized RMI structure

5.5.1 Lookup Request

After the client requests a remote reference, the RMI Agent has three options:

1. the reference points to a local object. The RMI Proxy makes a local RMI call and returns the result directly to the client.
2. the reference points to a remote object, but the reference has been requested already. The RMI Agent returns the stub extracted from its database.
3. otherwise, the RMI Agent sends a Lookup Request (LR) packet to the Proxy on the fixed side.

The format and the explanations of the symbols are the following:



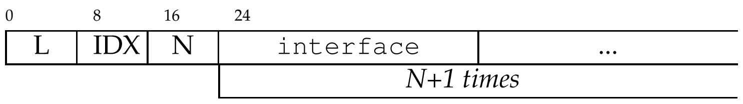
Symbol	Size	Explanation
L	1 octet	Lookup invocation
URL	variable	The remote reference

The first field tells the Proxy that the following is a lookup request, and URL is the reference of the remote object as listed in the remote Registry. In Java terms, the invocations done by the Proxy resembles the following:

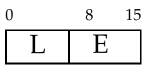
```
stub = Registry.Lookup(URL)
```

5.5.2 Lookup Answer

After receiving an LR message, the RMI Proxy invokes a `Registry.lookup(URL)` method and receives the RMI stub from the remote object. The Proxy then extracts information from the stub through Java `introspection` and saves the information in a database. Then it sends one of the following Lookup Answer (LA) messages to the RMI Agent:



or



The packet fields have the following meaning:

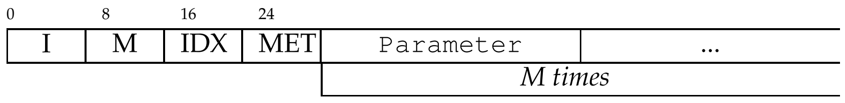
Symbol	Size	Explanation
L	1 octet	Lookup Invocation
IDX	1 octet	Database index
E	1 octet	Error code
N	1 octet	# of interfaces
`interface`	variable	Java Interface

Symbol	Possible value
E	`NOT BOUND`
	`REMOTE EXCEPTION`
	`CONNECTION REFUSED`

Once having received an LA packet, the RMI Agent checks if the first field contains an error code. In this case it throws an exception. Otherwise it uses the information to create the dynamic run-time stub (see Sec. 5.4.1) and it saves it to a database. The `IDX` value is the hash key to use for requesting the same object on the Proxy side.

5.5.3 Invocation Request

To execute a method in a remote object, the message that the RMI Agent sends to the Proxy is the Invocation Request(IR) message:



Where the fields have the following meanings:

Symbol	Size	Explanation
I	1 octet	Invocation
IDX	1 octet	Database index
M	1 octet	# of parameters
MET	1 octet	Method index
`Parameter`	variable	Java Object

The Proxy receives the message, retrieves the stub from its database using the `IDX` index and invokes the method indexed by `MET` with the `Parameter` (s). In Java terms, the invocation resembles the following:

```
IDX.MET(Parameter, ...)
```

5.5.4 Invocation Result

After the invocation is done, the RMI Proxy sends the result to the RMI Agent using the Invocation Result (IR) packet:

0	8	16
I	E	`Return Value`

with the following meanings:

Symbol	Size	Explanation
I	1 octet	Invocation
E	1 octet	Error code
Return	variable	Java Object

Symbol	Possible value
E	`OK`
	`REMOTE EXCEPTION`
	`STALE`

If the invocation was successful, the error code is `OK` and the return value is sent. Otherwise, the `E` field is filled with `STALE` if one of the indexes was stale (and a new lookup invocation is needed) or with `REMOTE EXCEPTION` if another exception was raised.

5.6 Performance Evaluation

To validate our solution and to check its performance, our implementation was measured in field trials. The objective was to study how our system behaves in different circumstances and to compare its performance with regular RMI.

5.6.1 Test Arrangements

In the measurements, we used the configuration specified in Table 5.2. The mobile node was connected to the Access Node using GSM HSCSD data service. Technically the HSCSD consists of two parts: multislot capability and the modified channel-coding scheme. The former provides the use of several parallel time slots per user where normal GSM can use only one. At the beginning of the HSCSD service, the maximum number of time slots is mostly limited to 3+1 (asymmetrical connection, three time slots for downlink) and 2+2 (symmetrical connection). The modified channel-coding scheme provides the user with a data rate of 14.4 kbps instead of the original maximum 9.6 kbps. In order to achieve higher bit rates, a more efficient puncturing method is used. This, on the other hand, decreases the radio interface error-correction performance. Therefore, the 14.4 kbps channel coding cannot be used when there is a lot of noise and interference affecting the quality of the radio signal. Depending on connection type and the capabilities of the network infrastructure, the maximum user data rate can be 28.8 kbps using a modem connection, 38.4 kbps using an ISDN V.110 protocol, or 57.6 kbps using an ISDN V.120 protocol.

Table 5.2: Hardware used in the measurements

Mobile Terminal	Toshiba Portégé 7020CT (Intel Pentium II/366MHz; 128Mb Main Memory)
Access Node	Compaq DeskPro EN6350 (Intel Pentium II/400MHz; 128Mb Main Memory)
GSM Phone	Nokia CardPhone 2.0 (Prototype)
Access Node Modem	Multitech MT2834ZDXI (28.8Kbps)

We performed the measurements using a modem connection, thus having 28.8 kbps as maximum user data rate. The following combinations of channel coding (CC) and time slots were used: 1+1 (96CC; original GSM), 1+1 (144CC), 2+2 (96CC), 2+2 (144CC), and 3+1 (96CC). The measurements were conducted in a normal office environment with good HSCSD radio link conditions.

The software package used in the test was the KaRMI benchmark suite provided by the Institute for Program Structures and Data Organization of the University of Karlsruhe [78][4]. The test cases used are shown in Table 5.3. Each test was repeated 20 times in every configuration.

Since the implementation of the Java Virtual Machine and the underlying TCP/IP implementation are different in different operating systems, we conduct all the experiments in a Windows environment and in a Linux environment, as described in Table 5.4.

5.7 Summary of Performance Results

5.7.1 Lookup results

In the lookup case, we measured the time to get the reference to a remote object. In our optimized implementation, we did not measure the case when the reference is found in the RMI Agent's reference cache; in this

[4]Software available at http://wwwipd.ira.uka.de/~hauma/EfficientRMI/

Table 5.3: Test cases set

Name	Parameters type	Return value
Void	null	null
ReturnPing	Byte[500]	Byte[500]
PingImage	Byte[5998]	null
PingText	Byte[9689]	null
ReturnText	Byte[9689]	Byte[9689]

Table 5.4: Test Environment

	Client	Server
Windows		
OS	Windows 98	Windows NT (SP 6)
JDK	Sun JDK 1.2.2	Sun JDK 1.2.2
Linux		
OS	Linux 2.2.14	Linux 2.2.14
JDK	Sun JDK 1.2.2	Sun JDK 1.2.2

case, the reference is found in a few milliseconds depending on the speed of the underlying hardware and Java implementation.

Table 5.5: Comparison between normal RMI and optimized RMI for the *Windows Lookup* case

	Original RMI				Monads			
	Min	Max	Median	Average	Min	Max	Median	Average
1+1 (96CC)	5540	6530	5685	5693	1260	4840	1290	1528
1+1 (144CC)	1000	8900	6565	6667	1260	1430	1320	1346
2+2 (96CC)	1543	6040	5135	5118	1150	1380	1210	1234
2+2 (144CC)	1071	8240	5710	5682	1160	2140	1260	1313
3+1 (96CC)	5100	8300	5210	5315	1100	2310	1210	2310

Table 5.6: Comparison between normal RMI and optimized RMI for the *Linux lookup* case

	Original RMI				Monads			
	Min	Max	Median	Average	Min	Max	Median	Average
1+1 (96CC)	6304	6478	6390	6388	1116	2346	1136	1200
1+1 (144CC)	6490	8332	6684	7080	1107	1622	1119	1154
2+2 (96CC)	5118	5745	5216	5263	958	1225	979	995
2+2 (144CC)	5382	8364	6214	6378	994	5878	1068	1625
3+1 (96CC)	5308	9547	5963	6144	961	1139	980	1139

Tables 5.5 and 5.6 show the results of lookup tests in Windows and Linux respectively. As expected, our implementation is significantly faster due to the reduction of unnecessary round-trips (see Figure 5.6). Using the original GSM (1+1, 96CC) our implementation is more than four times faster than the normal RMI in a Windows environment, and more than five times faster in Linux. An interesting result is that in HSCSD data service using the 14.4 kbps channel-coding scheme, the round-trip time is slightly higher than using the original 9.6 kbps channel coding. Since fewer round-trips are needed in our implementation, we gain more when 14.4 kbps is used as shown in Figure 5.7. The actual throughput does not significantly affect lookup results, since there is only a small number of bytes to send both in our implementation and in the original Java RMI.

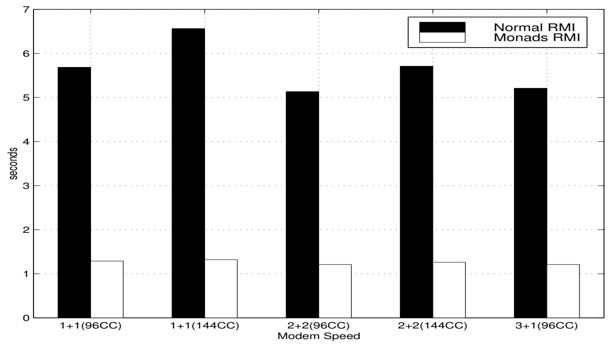


Figure 5.6: Summary of the lookup test in a Windows environment

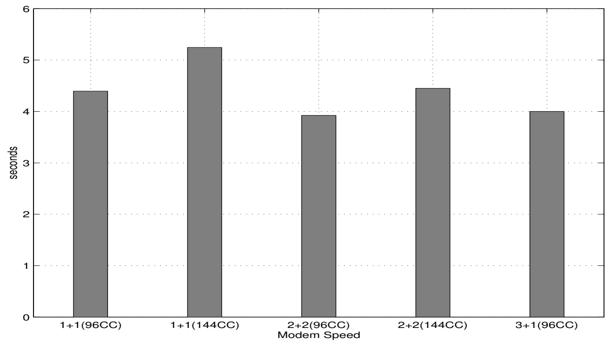


Figure 5.7: The difference between original RMI lookup and optimized lookup

5.7.2 Invocation results

In the Invocation case, we performed an extensive study using different kernels found in the KaRMI package. Table 5.3 summarizes the test cases we used.

First, to evaluate basic overhead caused by RMI, we used a simple remote method; with no parameters nor return value. The results are given in Tables 5.7 and 5.8 for Windows and Linux respectively. As there are only a few bytes to transfer in both directions, the difference between our implementation and original RMI is insignificant. In our implementation there is some additional processing overhead, as all invocations go through the RMI Agent. However, with the slow wireless communication path being the bottleneck, this is not a problem. In Figure 5.8, the results of the Windows environment are illustrated.

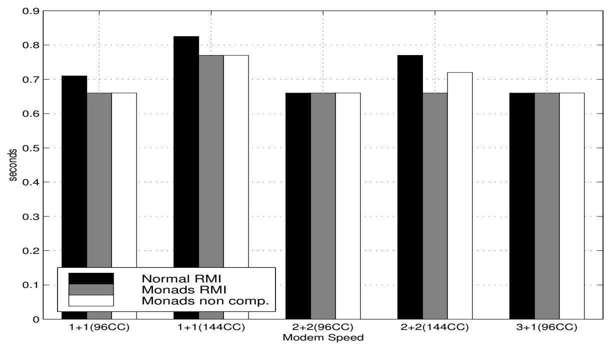
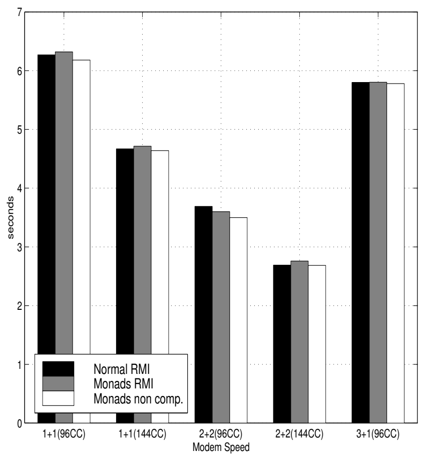


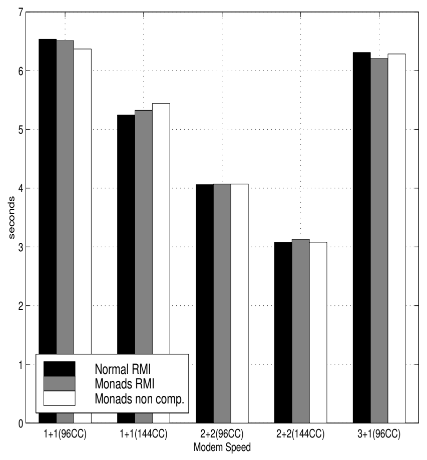
Figure 5.8: Summary of the void ping() test in a Windows environment

Next we evaluate the case where the remote object takes an array of bytes (500) as an argument, and as a return value, returns the same array back. Tables 5.7 and 5.8 summarizes the results of this test case in Windows and Linux respectively. In this case, there is now more data to send, and therefore using compression affects the results significantly. This is mainly due to the content of the array; in the KaRMI package all arrays are always initialized with zeros.

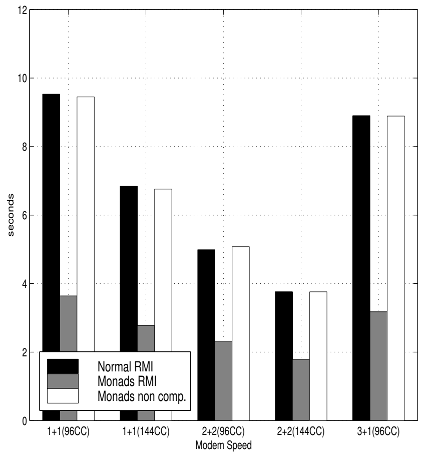
In order to evaluate our implementation with more realistic data, we used a case where the client sends a file to the server as a parameter of a remote method, and another case, where the server also returns the same file as a return value. As the content of the file affects the results of our implementation significantly when compression is used, we selected two files with very different content; the text file used was a HTML page of 9689 bytes and the image file used was a GIF image of 5998 bytes. The general purpose compression algorithm we are using is not able to compress GIF images. As a result, we do not gain anything from using compression with very random data, but on the other hand, the additional overhead needed in this case is not significant. When we are transferring text data, for example, the compression works well, and our implementation is significantly faster than the original RMI. The summaries of image and text transfer results are shown in Tables 5.7 and 5.8. The results are summarized in Figure 5.9.



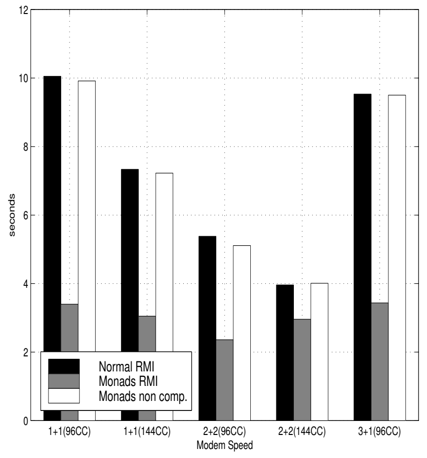
(a)



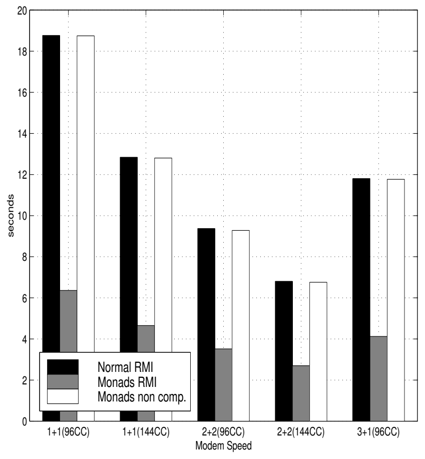
(d)



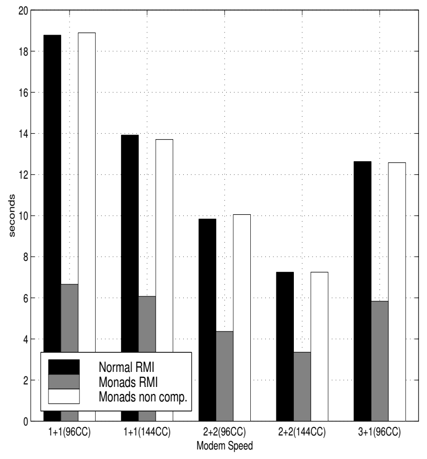
(b)



(e)



(c)



(f)

Figure 5.9: Figure (a) shows the invocation times of uplink image transfer using different link speeds in a Linux environment. Figure (b) shows the invocation times of uplink text transfer while Figure (c) shows the invocation times of two-way text transfer in a Linux environment. Figures (d),(e), and (f) show the corresponding times in the Windows environment

	Original RMI				Monads Compressed				Monads Uncompressed			
	Min	Max	Median	Average	Min	Max	Median	Average	Min	Max	Median	Average
	Comparison between normal RMI and optimized RMI for the *Windows Void* case											
1+1 (96CC)	650	1380	710	719	650	720	660	662	600	880	660	651
1+1 (144CC)	760	1980	825	867	710	2040	770	835	710	2030	770	826
2+2 (144CC)	710	1980	770	821	650	720	660	681	710	770	720	722
3+1 (96CC)	650	1370	660	697	600	660	660	654	650	660	660	659
	Comparison between normal RMI and optimized RMI for the *Windows ReturnPing* case											
1+1 (96CC)	1640	1710	1650	1672	710	820	770	749	1530	3400	1540	1689
1+1 (144CC)	1480	4060	1540	1649	820	1480	855	881	1420	5540	1455	1667
2+2 (144CC)	1100	1590	1150	1193	710	1210	770	804	1090	3790	1100	1301
3+1 (96CC)	1370	1540	1430	1437	650	2580	685	782	1310	3460	1320	1529
	Comparison between normal RMI and optimized RMI for the *Windows PingImage* case											
1+1 (96CC)	6260	13900	6535	7382	6150	15980	6510	7475	6150	16090	6370	7352
1+1 (144CC)	5210	33940	5245	8436	5220	11210	5325	5989	5160	8680	5440	6031
2+2 (144CC)	3020	4510	3075	3238	3070	4720	3130	3202	3020	6810	3080	3477
3+1 (96CC)	6200	16750	6310	7780	6150	12580	6205	6541	6150	10380	6285	6755
	Comparison between normal RMI and optimized RMI for the *Windows PingText* case											
1+1 (96CC)	9450	17300	10050	10488	3350	6590	3400	3576	9390	19230	9915	10840
1+1 (144CC)	7250	10270	7335	7563	2960	6420	3050	3298	7140	8240	7225	7296
2+2 (144CC)	3950	4330	3960	3990	2690	5380	2960	3125	3950	7960	4010	4399
3+1 (96CC)	9440	12360	9530	9897	3350	6260	3435	4248	9440	11640	9500	9625
	Comparison between normal RMI and optimized RMI for the *Windows ReturnText* case											
1+1 (96CC)	18340	19220	18780	18691	6090	10930	6150	6657	18290	27080	18895	19811
1+1 (144CC)	13680	26920	13920	15027	5000	10990	5110	6072	13620	15050	13705	13841
2+2 (144CC)	7190	9940	7250	7396	2910	6750	3075	3359	7200	10540	7250	7558
3+1 (96CC)	12570	13290	12630	12682	4450	7580	5820	5836	12520	15220	12580	12715

Table 5.7: Comparison between normal RMI and optimized RMI in the Windows environment

	Original RMI				Monads Compressed				Monads Uncompressed			
	Min	Max	Median	Average	Min	Max	Median	Average	Min	Max	Median	Average
	Comparison between normal RMI and optimized RMI for the *Linux Void* case											
1+1 (96CC)	749	1543	779	809	699	720	719	716	699	719	719	717
1+1 (144CC)	799	1663	799	844	759	2469	759	846	759	800	792	780
2+2 (144CC)	710	1953	719	781	669	1899	719	772	679	720	719	714
3+1 (96CC)	679	2744	694	891	639	1070	659	679	639	2016	649	847
	Comparison between normal RMI and optimized RMI for the *Linux ReturnPing* case											
1+1 (96CC)	1829	1879	1839	1846	800	868	800	803	1740	1837	1760	1756
1+1 (144CC)	1430	1507	1439	1446	790	898	800	805	1390	2210	1440	1466
2+2 (144CC)	1078	1119	1080	1089	720	1330	760	786	1110	1177	1120	1125
3+1 (96CC)	1319	1379	1350	1352	700	2560	720	939	1300	3430	1320	1495
	Comparison between normal RMI and optimized RMI for the *Linux PingImage* case											
1+1 (96CC)	6239	6340	6269	6276	6220	12660	6320	7126	6180	6241	6180	6191
1+1 (144CC)	4629	5829	4669	4826	4590	12510	4714	5690	4600	4791	4639	4652
2+2 (144CC)	2659	3189	2689	2742	2680	7200	2760	3170	2680	3200	2685	2771
3+1 (96CC)	5779	6859	5800	5890	5740	8430	5804	5938	5739	9100	5780	6056
	Comparison between normal RMI and optimized RMI for the *Linux PingText* case											
1+1 (96CC)	9479	10109	9529	9599	3460	5570	3480	3641	9420	9483	9450	9451
1+1 (144CC)	6789	7479	6840	6894	2720	3560	2720	2778	6750	7170	6760	6838
2+2 (144CC)	3712	5490	3764	3938	1719	3000	1720	1790	3750	3920	3760	3771
3+1 (96CC)	8879	9809	8900	8963	3160	3209	3180	3178	8840	10300	8890	9203
	Comparison between normal RMI and optimized RMI for the *Linux ReturnText* case											
1+1 (96CC)	18740	18858	18765	18774	6331	6970	6360	6391	18719	19880	18749	18854
1+1 (144CC)	12829	13450	12840	12920	4600	10820	4655	5595	12750	16830	12800	13079
2+2 (144CC)	6691	7131	6805	6816	2671	3920	2700	2762	6711	7190	6760	6806
3+1 (96CC)	11759	12463	11804	11870	4100	4188	4122	4126	11729	15151	11766	12190

Table 5.8: Comparison between normal RMI and optimized RMI in the Linux environment

5.8 Mobile RMI

The solution presented in the previous pages does not support mobility. If the client is located in a mobile host and the connection between it and the RMI Agent on the fixed side is interrupted, the RMI system could find itself in an unstable state until the timeouts in the RMI (and ωRMI) protocol expire. On the other end, ωRMI has been designed with this problem in mind, and it is not difficult to expand it to support mobility. In this section we give a description of ηRMI Protocol, that is, an extension of the ωRMI protocol to allow the mobility of the RMI client.

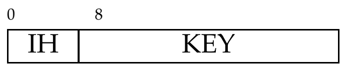
5.9 The Nomadic RMI Protocol (ηRMI)

5.9.1 The ηRMI Protocol Messages

This section gives a description of the different messages used in the ηRMI protocol. The description includes the format of the message and its fields.

5.9.1.1 The Init Handover (IH) message

This message is used to start a handover procedure. The format of the IH message is the following:

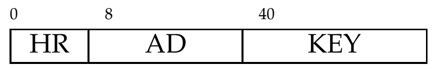


Symbol	Size	Explanation
IH	1 octet	Handover Procedure
KEY	variable	Client Identification Key

The identification key is used by the Proxy to recognize the identity of the client when it will reconnect in a different domain.

5.9.1.2 The Handover Ready (HR) message

This message announces the connection of the mobile host to a new RMI Proxy. The format of the HR message is the following:

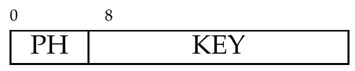


Symbol	Size	Explanation
HR	1 octet	Handover Ready
AD	4 octets	IP address
KEY	variable	Client Identification Key

AD is the address of the RMI Proxy to which the client was connected previously.

5.9.1.3 The Proxy Handover (PH) message

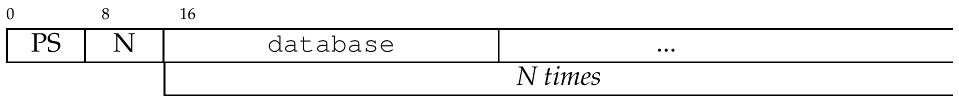
The PH message connects the receiver RMI Proxy with an old one. The format of the message is the following:



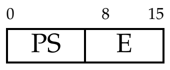
Symbol	Size	Explanation
PH	1 octet	Proxy Handover
KEY	variable	Client Identification Key

The KEY field is used to recognize (and authenticate) the client.

5.9.1.4 The Proxy Sync (PS) message



or



The packet fields have the following meaning:

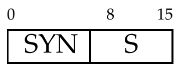
Symbol	Size	Explanation
PS	1 octet	Proxy Synch
E	1 octet	Error code
N	1 octet	# of databases
`database`	variable	Proxy database

Symbol	Possible value
E	`KEY UNKNOWN`
	`DATABASES EMPTY`
	`CONNECTION REFUSED`

With this message the databases of the mobile RMI Agent are transferred from the old Proxy to the new one.

5.9.1.5 The Synchronization Status (SYN) message

This message is delivered to the client when a synchronization between the old Proxy and the new one has begun. Its format is:

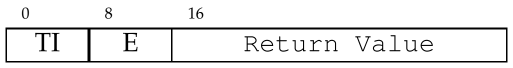


Symbol	Size	Explanation
SYN	1 octet	Synchronization Status
S	1 octet	Status

Symbol	Possible value
S	`SYNCHRONIZATION STARTED`
	`CONNECTION REFUSED`
	`SYNCHRONIZATION DONE`

5.9.1.6 The Tunnel Invocation (TI) message

This message transfers return values of pending requests to the new proxy. The format of the TI message is



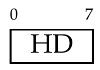
with the following meanings:

Symbol	Size	Explanation
TI	1 octet	Tunneled Invocation Answer
E	1 octet	Error code
Return	variable	Java Object

Symbol	Possible value
E	`OK`
	`REMOTE EXCEPTION`
	`STALE`

5.9.1.7 The Handover Done (HD) message

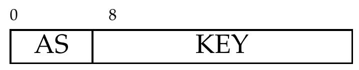
This message communicates to the new proxy that all the pending requests have been delivered. Its format is:



Symbol	Size	Explanation
HD	1 octet	Handover Done

5.9.1.8 The Abort Synchronization (AS) message

This message is used by a proxy to communicate to its peer that the synchronization needs to be aborted. The structure of the AS message is:



Symbol	Size	Explanation
AS	1 octet	Abort Synchronization
KEY	variable	Client Identification Key

The KEY field is used to identify the client.

5.9.2 Protocol Operations

Figure 5.10 depicts the sequence diagram for the ηRMI protocol.

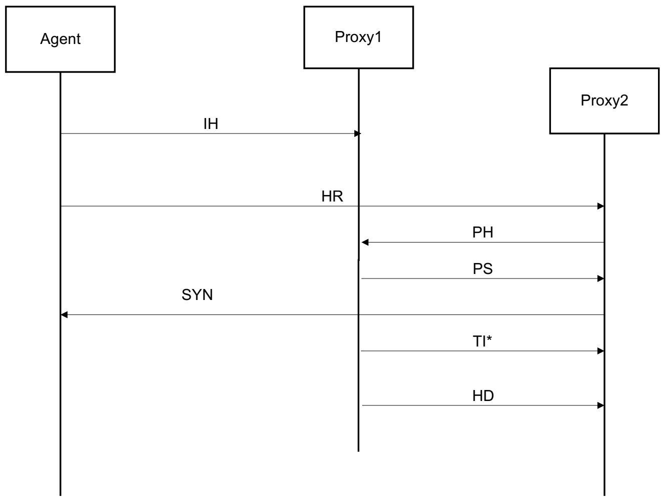


Figure 5.10: The Handover sequence diagram for ηRMI

When the host where the RMI client is located recognizes the need of a change in the access node[5] an Init Handover (IH) message is sent to the RMI Proxy to communicate the desire to start a handover procedure. The identification key is used by the Proxy to recognize the identity of the client when it will reconnect in a different domain. When the Proxy receives the IH message it saves all the information related to the mobile host to a database and buffers any data coming from the remote server for the client.

At this point the connections between the Agent and the Proxy can be interrupted and the mobile host can reach its new destination.

Note that the protocol works even if the disconnection is not voluntary. The RMI Proxy senses when the connection with a remote Agent is closed, and it can start a handover procedure by itself (see also Figure 5.11). If the disconnection was definitive, a timeout in the Proxy would release the memory reserved for that Agent.

When the mobile host connects to a new access node, it sends a Handover ready (HR) message to the new Proxy. The AD field is the IP address of the previous RMI Proxy and the KEY is the identification of the Agent. The length of this field depends on security requirement, since it can be used as

[5]This need could be triggered directly by an ηRMI-aware application.

an authorization token. In this way malicious entities cannot impersonate a client host and disrupt its services sending fake HR messages.

When the Proxy receives this message, it opens a tunnel to the well-known synchronization port of the old Proxy, and sends a Proxy Handover (PH) request. If the request is denied, the client will receive a PS message with the status `CONNECTION REFUSED`. The old Proxy, once it has received the PH message, checks if it knows the KEY. In this case it answers with the Proxy Sync (PS) message. If the KEY is unknown, it will return an error message. With this message the databases related to the mobile RMI Agent are transferred from the old Proxy to the new one. The new proxy sends a SYN message to the client. At this point the client knows that its databases are located at the new proxy. After a reconnection the Agent has to deliver the IP address of the new Proxy in the HR message.

If there are no pending requests, the old Proxy releases the stubs from the remote server and frees the memory. The new Proxy obtains new stubs from the remote servers as usual. On the other hand, if some servers have not yet answered all the client invocations there can be pending requests. When the remote server completes its tasks and returns the pending return values these are tunneled with the Tunnel Invocation (TI) packet from the old to the new Proxy Agent. When the old Proxy receives all the pending answers from the remote servers it was connected to it sends a Handover Done (HD) packet to the new Proxy and frees all the references it had with the clients.

At this point the tunnel between the two proxies is closed and the handover is completed.

5.9.3 Error Behaviour

While connections between Proxies can be considered reliable, the link between the client host and the access node can break. In fact, the client (or the device itself) can initiate a handover procedure while the previous one is not concluded yet. In this section we demonstrate how the protocol recovers in these cases.

If the IH message is lost because the host initiated a handover before the message reached the access node, nothing happens to the fixed side. As the client connect to a new Proxy, it sends the HR message to it and the protocol continues as usual. Figure 5.11 depicts the scenario. This situation is equivalent to an unvoluntary disconnection. The Proxy will recognize the loss of connection with the mobile host and start the handover procedure.

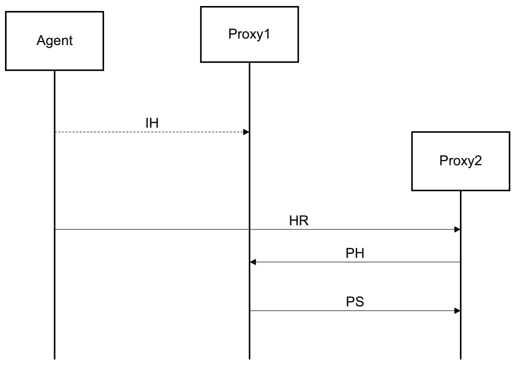


Figure 5.11: The protocol aborting an ongoing handover procedure

If the HR is delivered and then the client initiates another handover, it sends an IH message. The new Proxy sends an AS message to the old proxy and the synchronization is aborted. The new Proxy deletes all the information it stored about the client and the tunnel between the two proxies is closed (Fig 5.12).

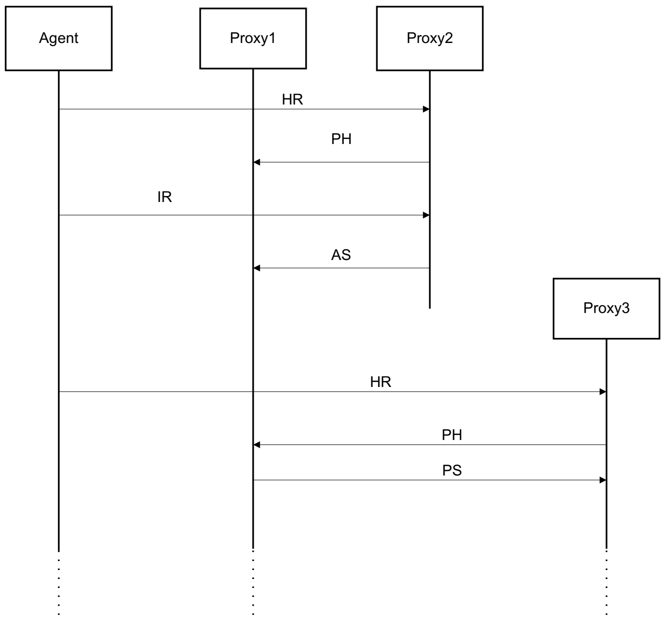


Figure 5.12: The protocol recovering from a loss of IH message

If the IH message is lost, when the client connects to another proxy it will send a HR message containing the IP address of the old proxy. The new proxy, following the protocol, will send a PH message to the old proxy. At this point, it will recognize that another handover took place, and it will send an AS message to the proxy with whom it had previously started a synchronization. The procedure then follows the protocol.

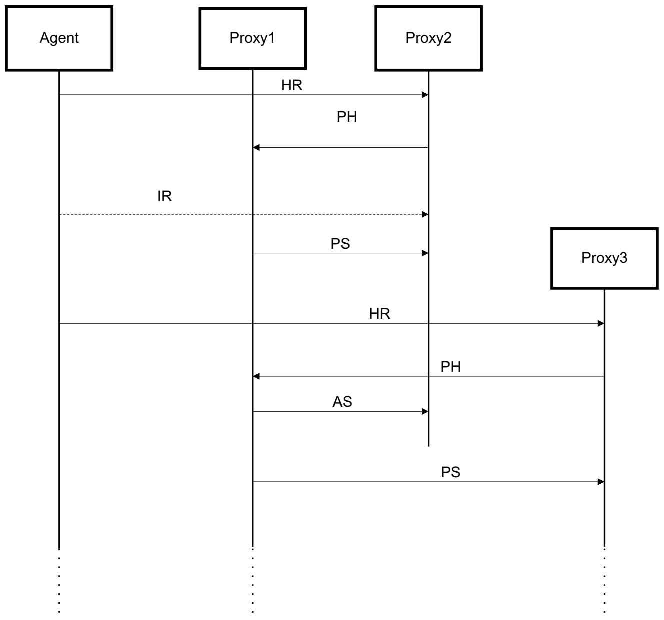


Figure 5.13: The protocol recovering from a loss of IH message during a second handover

Figure 5.14 describes a complex example of how the protocol works. A star indicates the arrival of a return value. In detail:

1. The Agent invokes a method (i1) to a remote server through Proxy1. Before the server answers, the client starts a handover procedure.
2. The Agent connects to Proxy2 through a new access node and completes the procedure since it receives the SYN message.
3. The return value of the i1 (r1) arrives to Proxy1 while the client invokes a new method (i2) through Proxy2. Before the result values are returned, the client initializes another handover.

4. The client connects to Proxy3. Proxy1 tunnels r1 to Proxy2 with the TI1 message. At the same time the return value of i2 (r2) is ready in Proxy2.
5. Proxy1 sends a HD message and concludes the handover procedure releasing the tunnel with Proxy2.
6. Proxy2 tunnels r1 to Proxy3 and the value is returned to the client. The same happens with r2, after Proxy2 sends the messages TI2 and HD to Proxy3 and closes the tunnel.
7. The handover is completed. From now on the client can invoke remote methods through Proxy3 until next handover.

5.10 Related Work

Optimizing Java and Java RMI performance have been quite popular research topics; see for example [1]. The work, however, has concentrated on high-speed networks—to the best of our knowledge there are no published results yet on Java RMI performance in slow wireless networks (cellular networks).

For the high-speed networks UKA serialization and KaRMI, developed at the University of Karlsruhe [78], provide a more efficient RMI for Java. The Manta project (Fast Parallel Java) in the Vrije Universiteit, Amsterdam [67, 96] has developed an efficient remote method invocation based on a transparent extension of Java for distributed environments. The Manta RMI is a part of the Manta environment and it cannot be used separately. At Indiana University there is a group that has conducted an interoperability and performance study of remote method invocation [12]. Another performance evaluation study has been carried out by the HORB project in Japan [51].

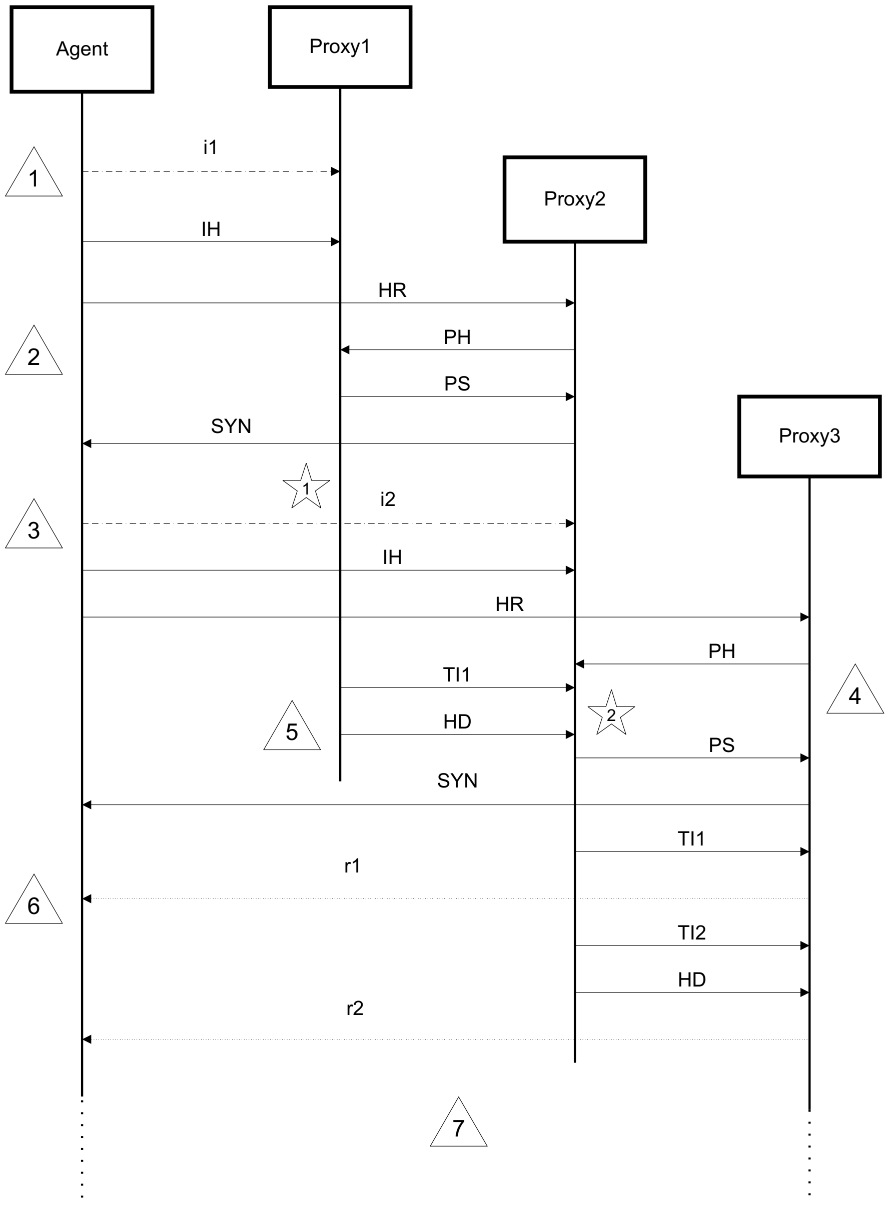


Figure 5.14: An example of the Nomadic RMI protocol.

Chapter 6

Middleware for Nomadic Applications

I love deadlines. I like the whooshing sound they make as they fly by.
- Douglas Adams

6.1 Introduction

In the previous chapters we described how a difficult task it is to try to adapt existing protocols and applications to mobile and wireless environments. The burden is often increased by the evolution of the "classic" computing paradigm that, ignoring the issues carried by the nomadic computing, drives toward high bandwidth demands and interactive always-connected technologies, including an exasperated use of multimedia.

On the other hand, sometimes this evolution introduces elements that can be reused in nomadic computing. For instance, in the previous chapter, describing our wireless version of Java RMI, we adopted the use of mediators. Mediators are part of a more general layer called "middleware". In this chapter we describe some middleware architectures that can be used by nomadic applications, and we show how middleware is evolving toward the concept of pervasive computing.

6.2 Middleware

A definition of “middleware” globally accepted by the academic society and the enterprise world does not exist. In general, the concept of middleware assumes a functional layer between the client and server. In the prototype implementation presented in the previous chapter the middleware layer was situated between the client and the network, but more generally it may provide service such as location and alias resolution, authentication and transaction semantics. Other behaviors associated with middleware include time synchronization and translation between data formats.

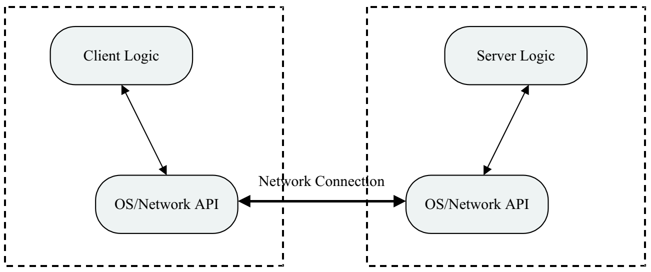


Figure 6.1: Middleware Architecture

This additional layer allows clients to interact with a generic abstraction of a service rather than with a specific process. Various services are provided through abstracted layers as well, blurring the distinction between services provided by the middleware and functionality added by servers. These abstractions allow applications to be developed to a standardized API without knowledge of the location or implementation of external functionality. This implementation hiding is one of the middleware model’s strengths, although it makes it difficult for the client to determine what performance it can expect from any given logic implementation.

6.3 Service Advertisement and Discovery

6.3.1 Introduction

As we will see later, the recent trends in mobile and ubiquitous computing created a variety of new protocols aiming to provide automatic “discovery” and configuration of devices and services. Unfortunately, the terminology in this area is not standardized yet, and it is prone to confusion. We will use the term “Lookup” and “Discovery” as follows:

Lookup – We use this term when we refer to a process of *locating* a resource. Arguments of a lookup operation may be an address or an exact name. Lookup is a *passive* operation and requires the existence of other services (directory or agent) to answer the request. Examples of lookup service are DNS and CORBA Naming Service.

Discovery – This is a more spontaneous process, in which *entities* discover other entities on the network, and present themselves to other entities. A discovery process can be used for a lookup process, but not vice-versa. Usually discovery requests no human administrative intervention.

The followings are some of the most well known discovery protocols available on the market or in a later stage of development.

6.3.2 Bluetooth

The Bluetooth [15] wireless technology was created to solve a simple problem: replace the cables used on mobile devices with radio frequency waves (Figure 6.2). Initially, the technology will be used as replacement for point-to-point cables, but solutions for forming personal area networks of Bluetooth devices will evolve later. Bluetooth is a low-power, short-range, wireless radio system. The radio has a range of ten meters and provides up to seven 1 Mb/s links to other Bluetooth devices.

Bluetooth channels use a frequency-hop/time-division-duplex (FH/TDD) scheme. The channel is divided into 625 μs intervals, called slots. A different hop frequency is used for each slot. The nominal link range is from 10 centimeters to 10 meters, but it can be extended to more than 100 meters by increasing the transmission power. Bluetooth can support an asynchronous data channel, up to three simultaneous synchronous voice channels, or a channel supporting simultaneously asynchronous data and synchronous voice. Each voice channel supports 64 kb/s synchronous (voice) link. The asynchronous channel can support an asymmetric link of maximally 721 kb/s in either direction while permitting 57.6 kb/s in the return direction, or a 432.6 kb/s symmetric link

The Bluetooth units can create both point-to-point and point-to-multipoint connections. A connection with two or several (maximum eight) units is called a *piconet* where all units are following the same frequency-hop scheme. To avoid interference between units, one of them automatically

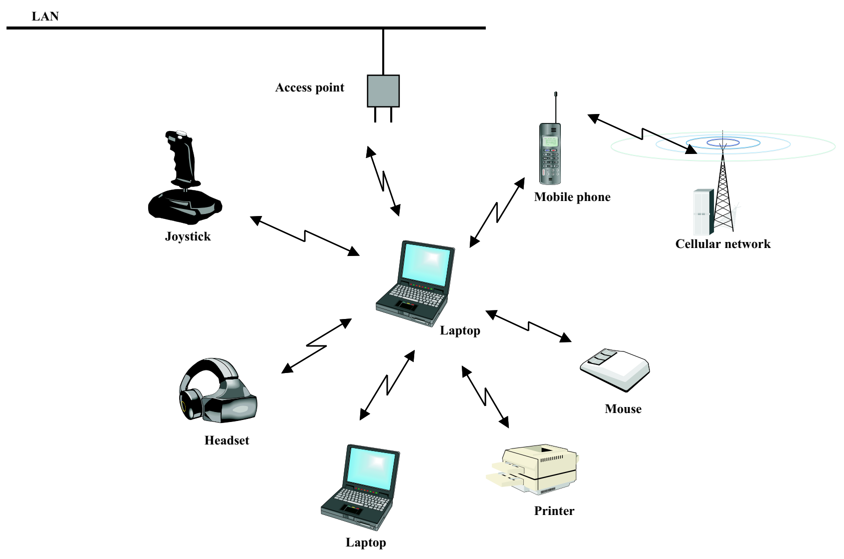


Figure 6.2: A Bluetooth scenario

becomes a master of the piconet. Units in a piconet can communicate with each other, share services and synchronize data. Individual devices can participate in more than one piconet at a time and can be in one of several states:

Standby the device is conserving power and waiting to connect to another Bluetooth device.

Inquire the device is *searching* for nearby Bluetooth devices.

Page the device is *connecting* to another Bluetooth device.

Connected the device is *connected* to another Bluetooth device.

Hold and park the device is participating in a piconet with varying degrees of power savings.

Two or several piconets can communicate with each other and are then called a *scatternet* (Fig. 6.3).

Bluetooth provides a simple API for enumerating the devices in range and browsing available services. Client applications use this API to search for

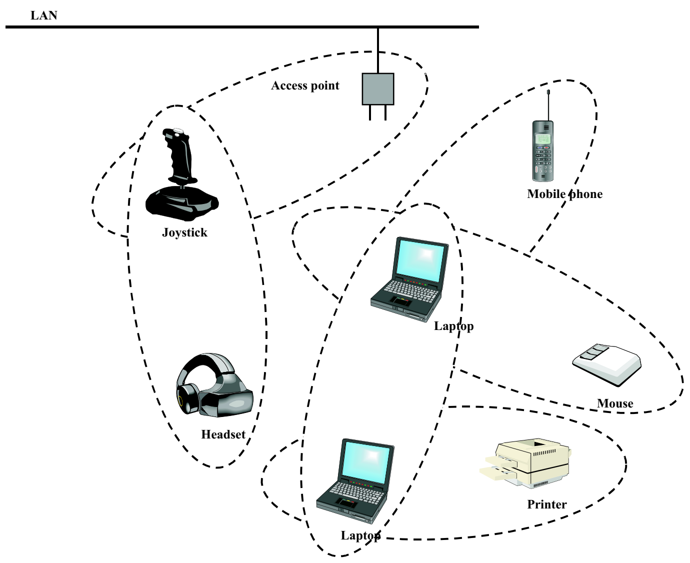


Figure 6.3: A scatternet of six piconets

available services either by service classes, which uniquely identify types of devices (such as printers or storage devices), or by matching attributes (such as a model number or supported protocol).

Since the range of Bluetooth is not impressing, it would be possible to listen to other units from a nearby room. To avoid such risks and to maintain privacy the Bluetooth physical layer includes authentication and encryption.

6.3.3 Jini

Jini [6] is an environment for spontaneous federations of services. It is based on Java and allows different Jini-enabled devices to announce the services they provide so that they can be found and used by other services in other devices. This provides a mean for spontaneous connections between different devices so that they can collaborate to carry out tasks that require multiple devices or services. For example a digital camera can join a local wireless network, discover printing or storage facilities available on the network, load the required software and use it to print or store images.

Jini technology consists of a programming model and a runtime infrastructure. The purpose of the programming model is to help build robust distributed applications by using a federation of services and client programs. It instructs the developers on how to design and implement applications for Jini and within the Jini environment. The runtime infrastructure is the environment where these applications run. It provides means and tools for adding, searching, contacting and employing services and resources available on the network.

Jini-enabled devices can discover a Jini environment, join it, look up other services and employ other services to carry out tasks.

6.3.3.1 Jini Services

The Jini infrastructure (Figure 6.4) operates on a relatively high level of abstraction and is not concerned with communication details. The RMI mechanism hides communication implementation details below a simple object lookup and remote class loading mechanism.

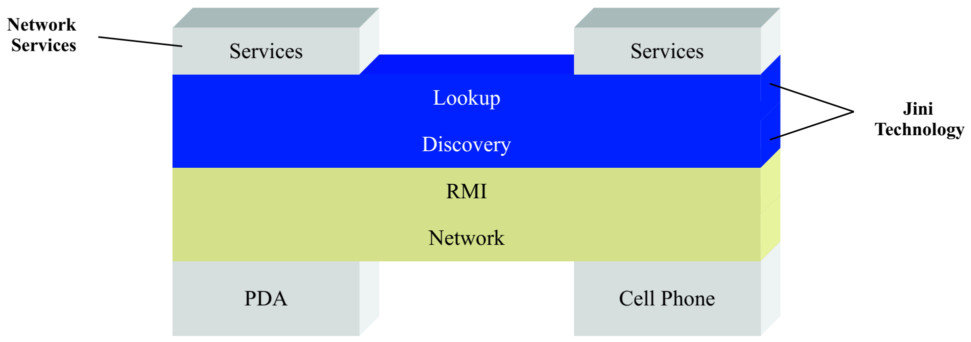


Figure 6.4: Jini Services on top of RMI

6.3.3.2 The Discovery Process

On top of RMI Jini provides the lookup service, which is accessed by a process called discovery and join. The lookup service keeps track of the services and records them in groups. A single service can belong to several groups. There may be one or more lookup services operating on a network but at least one lookup service must be available in the network.

As soon as a Jini-enabled device discovers that it has joined a new network it broadcasts a “presence announcement” message. This announcement contains information about how the service can be contacted and a list of groups it wants to join.

6.3.3.3 The Join Process

Once a device has discovered a lookup service it joins the community by sending information about itself. It does so through the stub it received from the lookup service. The information is sent by transmitting an object, which contains the service interfaces that the object wants to be made available. Once the lookup service has received this information it stores it in an internal database for future lookups and the join process is complete (Fig. 6.5).

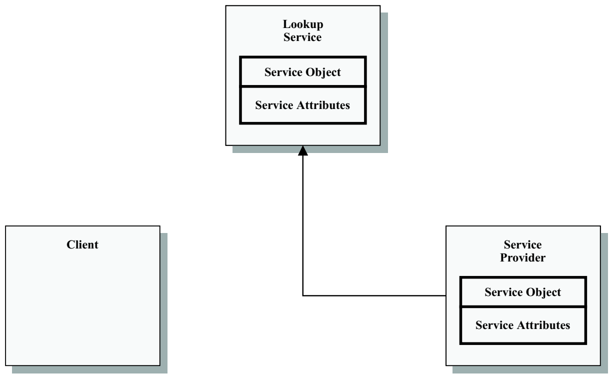


Figure 6.5: The join process

The services that the device provides are identified by the type of service interfaces it implements. Each kind of service is associated with one Java-based interface. Furthermore, the object contains service attributes that can be used to describe the service in more detail.

The lookup service stores and locates services based on the types of their interfaces and later clients interact with the service by invoking methods on an object that implements a certain interface. In addition to the Jini-specified service interfaces the device can make available applets, other attributes and objects that implement specific protocols for accessing its services.

6.3.3.4 The Lookup Process

Once a service has joined at least one group on the lookup service it is available to clients.

A client starts the search by locating and querying a lookup service for services of a certain type. The type is specified as Java interface. The lookup service uses its database to find matches to the query and returns the objects that match the query to the client (Fig. 6.6).

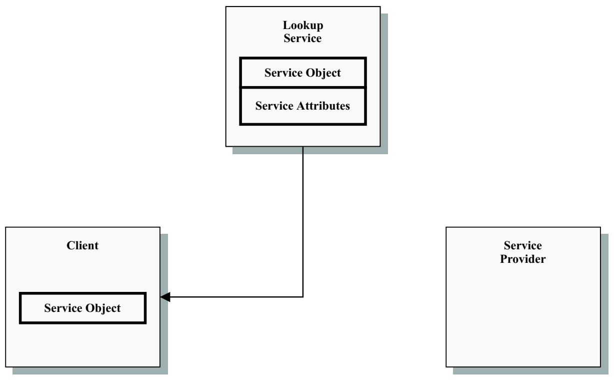


Figure 6.6: The lookup process

The client receives from the lookup service stubs of the searched services. Thought the stubs the client can interact with the remote device/service by invoking methods on the object implementing the service interface (Fig. 6.7).

In addition to the service interface the client can use the Java reflection mechanism to investigate the methods implemented by the object.

While the use of RMI is suggested by the tools made available in the Jini runtime environment, it is not mandatory. The role of Jini is to provide a way for the client to find the services it needs and to bootstrap the communication process. After the lookup service sends the client the service object, it is up to the client and the loaded object to take care of communications between the client and the service. The communications channel can be practically anything that can be implemented in Java code that is running on the client.

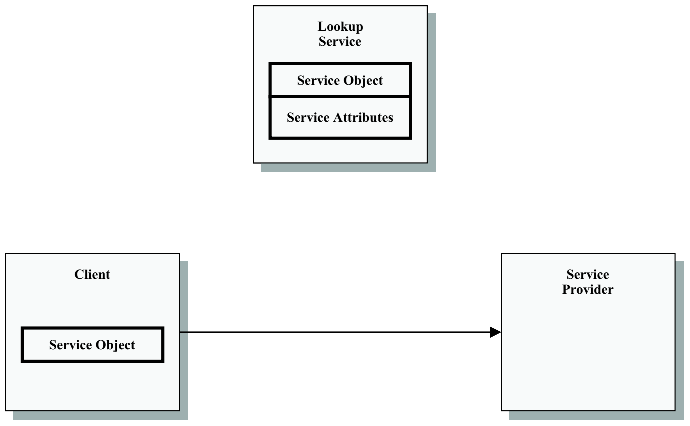


Figure 6.7: Service usage

6.3.3.5 Jini Reliability

The operational environment of Jini is inherently unreliable. Communication disruptions and the intermittent connection of the devices make programming reliable distributed applications a difficult task. The Jini technology programming model offers a small set of APIs that can help in creating reliable distributed systems. It approaches the problem using leasing. A lease is a grant of guaranteed access to a remote reference such as an object's limited to a specified period of time. It is the referencing object responsibility to renew the lease before it expires. Once a lease expires the garbage collector assumes that there are no longer remote references to it and its memory space can be recollected.

Furthermore, the Jini distributed event mechanism extends the Java event model, which works within a single Java virtual machine, to distributed systems. An object can register itself as a listener for events generated by a remote object. Once the event source has fired an event the Jini environment will take care of migrating the event to the registered listener objects. This infrastructure can be used extensively in the client to service communication.

6.3.3.6 Discussion

The Jini infrastructure and programming model are both based strongly on Java. The communication mechanism is RMI (see Sec. 3.3). RMI's object migration capabilities enable not only mobile data but also mobile code — Jini-enabled devices can send out Java code that executes on the client that wants to use the device. This is the most powerful mechanism in Jini that stems directly from Java. A device can provide as much functionality on the client using Jini as is needed and nothing apart from the basic Java runtime environment and Jini environment need be installed beforehand. Thus Jini truly enables spontaneous, complex interconnections between different devices with practically no prearrangements.

Unfortunately, as described in great detail in the previous chapter, RMI is not meant to be used in a wireless environment. Thus, most of these interconnections, at the time being, must be considered as not mobile.

The security implications of Jini are serious. If even simple devices such as light switches, locks and digital cameras are available as networked devices there has to be heavy security measures in place to prevent illegitimate use. Jini is aimed at making cooperating computing and physical resources easy to create, maintain and use. Thus security should be strong but easily managed so that the ease does not disappear due to too much trouble with security.

6.3.4 Salutation

Salutation is an architecture for service discovery under development by the Salutation Consortium [87]. The salutation architecture defines an abstract model with three Network Entities: Clients, Services and Salutation Managers (SM) as shown in the figure below.

Services register their capabilities with an SM, and clients query the SM when they need a service. After discovering a desired service, clients are able to request the utilization of the service through the SM. As depicted in Figure 6.9 Salutation defines its protocol based on RPC (see Section 3.2).

The SM manages all communication, and bridges across different communication media as needed through three different protocols:

- The Salutation Manager may set up the data pipe and then step into the background, allowing the Client and Service to manage the mes-

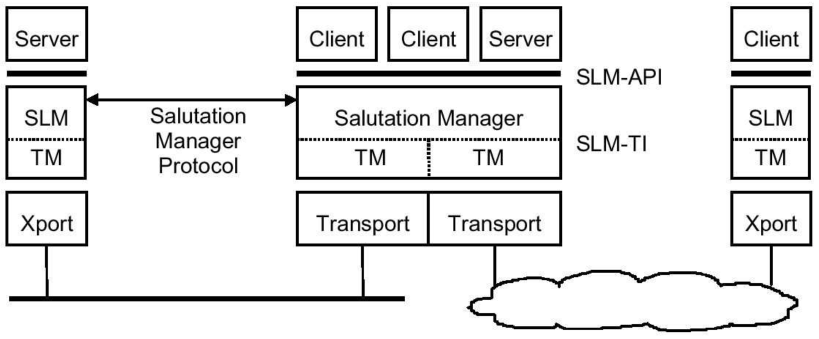


Figure 6.8: Model of The Salutation Manager

sage stream and data formats. This is known as **Native Personality**. In this personality the Salutation Manager acts as a Service Broker, and the applications, services and devices manage the interactions between Clients and discovered Services. Messages are exchanged between Clients and Services directly, without the involvement of the Salutation Manager.

- The Salutation Manager may set up the data pipe and manage the message stream, while the data formats are selected and controlled by the Client and Service. This is known as **Emulated Personality**. This personality is useful when a common messaging protocol does not exist between a Client and a discovered Service. All Messages under an Emulated Personality Protocol are carried by the Salutation Manager Protocol. Message exchange is native data in Salutation packets. Under the Emulated Personality Protocol, Client Messages go through Salutation Managers, however the Salutation Manager never inspects the contents or semantics of Messages.

- The Salutation Manager may set up the data pipe, manage the message stream, and provide the data format definition for Client/Service interaction. This is known as **Salutation Personality**. This personality provides a common messaging protocol and common data format between a Client and a discovered Service. Under the Salutation Personality Protocol, the message format and exchange protocol are defined by the Salutation Architecture and all messages are carried by the Salutation Manager Protocol.

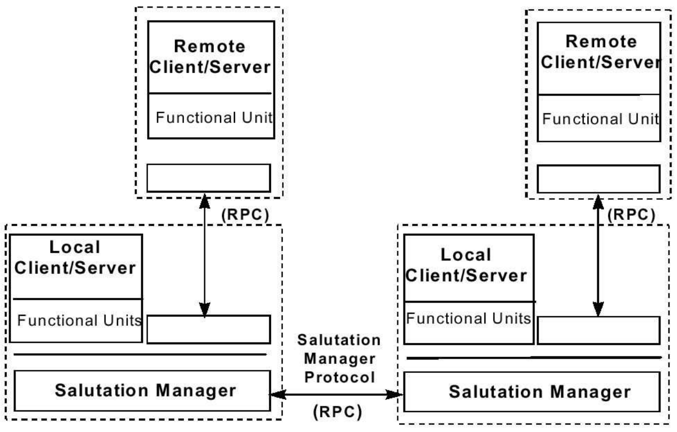


Figure 6.9: Use of RPC in Salutation

Salutation defines a specific (extensible) record format for describing and locating services. This format includes service type (such as [PRINT]) and attributes (such as "color"). Clients locate services by sending service requests that may include matching-binding attributes.

6.3.5 SLP

Service Location Protocol (SLP) is an IETF protocol for service discovery and advertisement [45]. Unlike Jini, Salutation, and UPnP, which all aspire to some degree of transport-level independence, SLP is designed solely for IP-based networks. SLP comprises three entities: service agents (SAs), user agents (UAs), and directory agents (DAs) (Fig. 6.11). SAs advertise the location and attributes of available services, while UAs discover the location and attributes of services needed by client software. UAs can discover services by issuing a directory-like query to the network. DAs cache information about available services. Unlike Jini, SLP can operate without directory servers, but the presence of DAs can substantially improve performance, by reducing the number of multicast messages and the amount of network bandwidth used. In fact, if DHCP is used to configure SLP agents with the location of DAs, then multicast is completely unnecessary. On the contrary, in the absence of DAs, UAs multicast requests for service and receive unicast responses directly from the SAs that control matching services. This tends to increase bandwidth consumption, but provides a simpler model, appropriate for small networks (such as a home LAN).

SLP has several mechanisms for discovering DAs: In passive discovery,

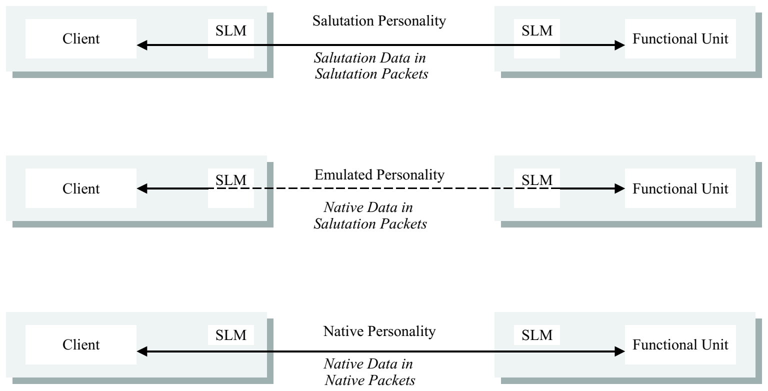


Figure 6.10: Personality Alternatives

SAs and UAs listen for multicast announcements from DAs, which periodically repeat these advertisements. In active discovery, SAs and UAs multicast SLP requests or use DHCP to discover DAs. When a DA is present, SAs and UAs use unicast communication to register their services and find appropriate services respectively.

SLP services are advertised through a service URL, which contains all information necessary to contact a service. Clients use the service URL to connect to the service. The protocol used between the client and server is outside the scope of the SLP specification, so its security model concentrates on preventing the malicious propagation of false information about service locations. SAs can include digital signatures when registering so DAs and UAs can verify their identity. Digital signatures can also be required when DAs advertise their availability, allowing UAs and SAs to avoid rogue DAs (that is, those without a proper signature).

6.3.6 UPnP

UPnP [95] is a proposed architecture for service advertisement and discovery supported by the UPnP Forum, headed by Microsoft. Unlike Jini, which depends on mobile code, UPnP aims to standardize the protocols used by devices to communicate, using XML. UPnP's device model is hierarchical. In a compound device (for example, a VCR/TV combo), the root device is discoverable, and a client (called a control point) can address the individual subdevices (for example, a tuner) independently.

As in JINI, UPnP has a multi-stage protocol. At the base, UPnP provides

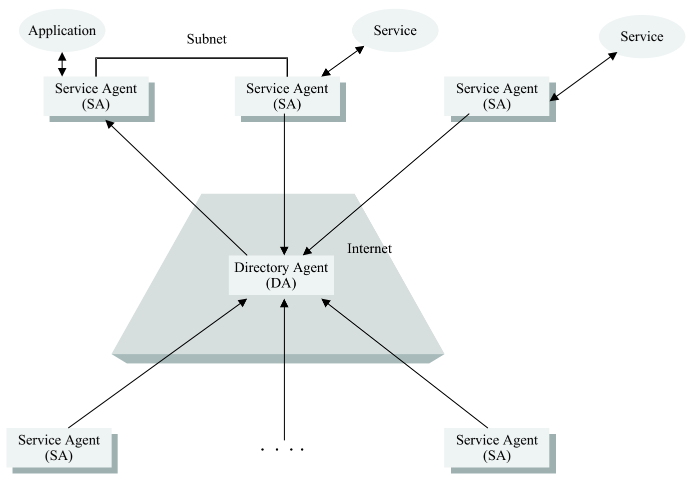


Figure 6.11: SLP Architecture

simple discovery, in which network addresses are discovered. Advertisement is done by a local broadcast announcement. When successful, the simple discovery call returns an IP address or URL plus a *device type*. Services are described by extended URLs, similar to (but completely incompatible with) SLP. The URL points to an XML file with an elaborate description of the device. Starting with this URL, UPnP defines a Web-based discovery protocol, which uses HTTP (with extensions). A UPnP device is said to export one or more *services*. Services are described in XML, and the XML can be a complete abstract description of the type of service, the interface to a specific instance of the service, and even the on-going (virtual) state of the service. The interface and state descriptions are intended to allow clients to implement custom interfaces to devices, by mapping local displays and operations to the abstract state and interface represented in the XML.

UPnP requires IP, not to mention HTTP and XML. Non-IP networks and interconnects can be bridged, at least at the level of the XML. UPnP has no specific security features. It depends on the network and Web infrastructure for its security. Thus, security is clearly an optional.

6.4 Pervasive Computing

6.4.1 Introduction

The term "Pervasive Computing" means different things to different people. For same, pervasive computing is about mobile data access. Others put the emphasis on "smart" and "proactive" spaces. Satyanarayanan [89] characterizes a pervasive computing environment as one saturated with computing and communication capability, yet so gracefully integrated with users that it becomes a "technology that disappears". The presence of computing power and communication devices are hidden from the end-user to whom is offered at the same time a powerful service-centric system.

In this section we present an overview of the main projects focusing on pervasive computing.

6.4.2 The Portolano Project

The University of Washington is carrying out a project called Portolano [84, 35]. In their vision of the new century, computing devices will be highly specialized to particular tasks, will be consumer items with a vast distribution, and their user interfaces will be invisible to all but the most sophisticated end-users.

To realize their vision the project focuses on an infrastructure that moves from system architectures that are vertically integrated (aiming to provide entire solutions to a problem) to horizontally layered architectures. Particular stress is given to the development of user interfaces able to handle different devices and that rely upon the user intent, and to the need of data-centric networks. The researchers expect to encounter several challenges in their effort, including resource discovery, intermittent connectivity and power consumption.

6.4.3 Oxygen

Oxygen [71] is a major research project of the Laboratory of Computer Science of the M.I.T. The Oxygen project vision predicts a future were computation will be freely available everywhere. Devices will lose their "anonymity", but they will personalize themselves in the user presence by finding whatever information or service he or she needs. The communication will not be carried out by clicking or typing, but simply naturally using speech.

To realize this futuristic vision Oxigen is focusing on several key technologies:

- At the heart of the system there is *Handy 21* similar to a cellular phone, but with an additional visual display, a camera, infrared detectors and a computer. Since it is all software-configurable it can change, for example, from a cell-phone to a plain FM radio.
- The second key technology of Oxygen is the *Enviro21*. This device stays attached to the environment (built into walls) and does the same services as the Handy21 but with greater speed and capacity.
- The Handy21 and the Enviro21 are linked by a novel network, called *Net21*. Its principal function is to create a secure collaborative region among Oxygen users.

Other enabling technologies in Oxygen consist of speech recognition, intelligent knowledge access and automation of every-day tasks through collaboration technology.

6.4.4 Endeavour Expedition

The Endeavour Expedition at the University of California in Berkeley [33] is a collection of projects that examines various aspects of ubiquitous computing. The goal is to enhance human understanding through the use of information technology, by making it dramatically more convenient for people to interact with information, devices and other people. A revolutionary Information Utility, which is able to operate at a planetary scale, will be developed. The underlying applications, which are used to validate the approach, are rapid decision making and learning. In addition, new methodologies will be developed for the construction and administration of systems of this unprecedented scale and complexity.

Of particular interest in the context of our discussion are:

Ninja –"Enabling Internet-scale Services from Arbitrary Small Devices" that develops a software infrastructures to support scalable, fault-tolerant and highly-available Internet-based applications [41].

Iceberg – An Internet-core Network Architecture for Integrated Communications that is seeking to meet the challenge for the converged network of diverse access technologies with an open and composable service architecture founded on Internet-based standards for flow routing and agent deployment [99].

6.4.5 MosquitoNet

The Mobile Computing Group at Stanford University (MosquitoNet) [75] has developed the Mobile People Architecture (MPA) [68] that addresses the challenge of finding people and communicating with them personally, as opposed to communicating merely with their possibly inaccessible machines. In their vision people should be reached regardless of the communication devices or applications they choose to use. They should be able to receive messages anywhere, but without revealing their whereabouts to anyone. Finally, users should be able to have all their incoming communications prioritized and filtered on their behalf to avoid unwanted messages such as “spams”.

The MPA architecture introduces the concept of routing between people by using the Personal Proxy. The proxy has a dual role: as a Tracking Agent, the proxy maintains the list of devices or applications through which a person is currently accessible; as a Dispatcher, the proxy directs communications and uses Application Drivers to marshal communication bits into a format that the recipient can see immediately. It does all this while protecting the location privacy of the recipient from the message sender and allowing the easy integration of new application protocols.

6.4.6 PIMA

The PIMA project [11] at the IBM T.J. Watson Research Center has developed a new application model for pervasive computing. In their view, devices need to be perceived as portals into the application and data space supported by the environment, rather than repositories of custom software. Furthermore, applications need to be seen as tasks performed on the behalf of a user, not as programs written to exploit the resources of a specific device.

Based on this vision they suggest a new application model characterized by a device-independent application development process, which includes abstract specification of the application front-end and requirements. The model should also support application discovery and resource negotiation at load-time. The run-time system should allow the resources to be dynamically shared among client devices and servers.

Part IV

Dynamic Nomadic-Aware Applications

Chapter 7

From Adaptation to Native Support

When you look long into an abyss, the abyss also looks into you.
- Friedrich Nietzsche

The previous chapters of this dissertation have been focused on the difficulties to port "classical" applications to a nomadic environment. One of the contributions of this research has been to show how all the different communication layers up to the middleware, need to be modified. While we suggested solutions for the adaptation, none of them is optimal and can be used to "nomadicize" all the possible existing applications and services.

On the other hand, the presence of computing devices, computer networks and wireless communication is increasing enormously in everyday life. Organizers start to be wirelessly connected, smart phones are becoming popular not only in Europe but they are spreading through the world, from the USA to China, from India to Africa, even if with a different speed of adoption. Furthermore, computing power in the form of processors are embedded in an increasing number of appliances, from cars to TVs, from microwave ovens to air conditioners.

This *ubiquitous* presence of the computing power is pushing the research of solutions for exciting new environments, where people interacts with embedded, invisible computers. It is not surprising to read in market analysis that future services and applications will be more and more intended for Pervasive Computing (see Sec. 6.4) which is adding more challenges to the ones already presented in Chapter 2, such as collaborative environments, information presentation, user interfaces and e-commerce. Many research groups in universities and companies are proposing several solutions to improve state-of-the-art Pervasive Computing, but still they fall into the category of adaptation.

In the following chapters we will try to give a suggestion for a different approach, in which the the adaptation is switched from the environment to the application. The main contribution will be the idea that applications and services should be created with a *genetic imprinting* so that they can dynamically modify themselves to adapt to the challenges of nomadic environments. A new class of applications that are fully aware of their surroundings and that take maximum advantage from the pervasive computing power and the enabling middleware solutions presented in Chapter 6.

This idea is illustrated in Chapter 9, while in next chapter we focus on the Telecommunication field and we show how it is possible to enable personal mobility with the help of agent technology. As we will see, this scenario poses the seeds of the evolution in the Dynamic Composition of Execution Environment.

Chapter 8

Agents in Personal Mobility

All profoundly original work looks ugly at first.
- Clement Greenberg

8.1 Introduction

As the first step we introduce an example of an alternative architecture to enable personal mobility in a telecommunication environment through the use of mobile agents. This basic architecture will introduce some intuitions that will be used and expanded in the next chapter, when we will introduce a new paradigm for constructing applications in a nomadic-aware manner.

8.1.1 Basic Elements

Below we present the basic definition that will be used in the following examples. All together they represent the minimum set of elements that compound the proposed architecture[1]. These basic elements are:

The User is the one who wants to start a service session.

The Service Provider is the stakeholder that offers services.

The Connectivity Provider is the stakeholder that takes care of physical connections between terminals and Service Providers.

[1]Even if some definitions remind of the TINA concept, this architecture does not rely on TINA.

The Terminal logically connects the User with the Service Provider.

Mobile Agents are software objects that act on behalf of the User through various Service Providers,

User Profile contains all the useful information about the User's behavior and preferences.

These basic elements are sufficient to build and support most of the scenario existing in personal mobility. In the next section, through some examples, we will refine the previous definitions.

8.2 Telecommunications Scenarios

A User has a subscription for a service in the local Service Provider. The type of service requires that the User Profile is located near the Service Provider as in Figure 8.1. This is due to the fact that most of the time the User will use a Terminal located in the Service Provider's domain.

When the association with the Service Provider is fixed (due to administrative or economical reasons) the Service Provider is called *Home Service Provider*. In this example the user is connected with his Home Service Provider and can use the Services provided by the Service Provider directly.

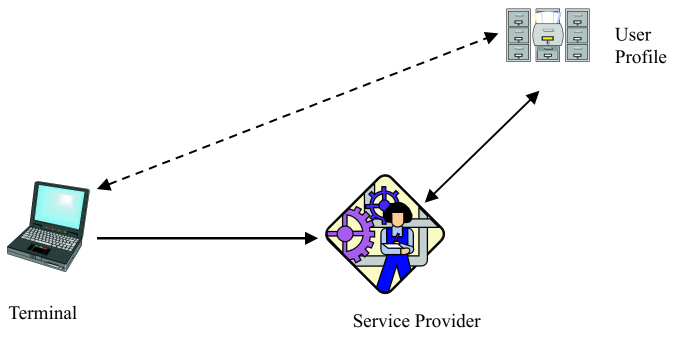


Figure 8.1: A normal subscription

When the user moves outside the domain of the Home Service Provider, he can use a Terminal connected with another Service Provider that has a *Federation Contract* with his Home Service Provider. A Service Provider different from the Home Service Provider is called *Visited Service Provider*. It does not contain the User Profiler, so the service cannot be allowed immediately. Anyway, since there exists a Federation of Service Providers and the User has a subscription with a Federated Service Provider, the User can obtain the service through the new Service Provider (see Figure 8.2).

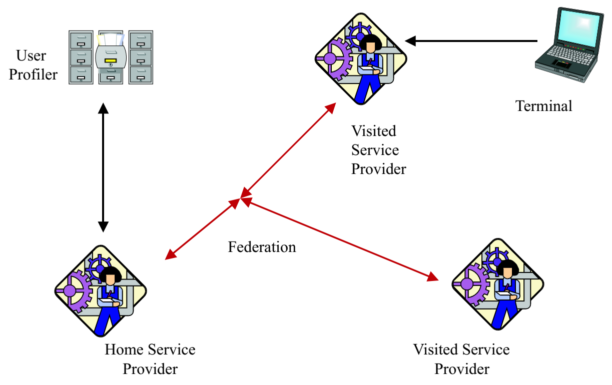


Figure 8.2: The user roams

To do that, a Mobile Agent connects to the Home Service Provider and gets the needed information from the User Profile and comes back to the Visited Service Provider (Figure 8.4).

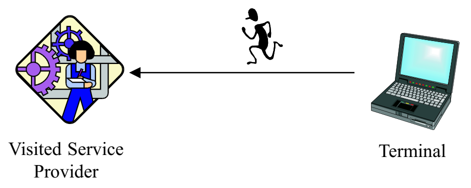


Figure 8.3: The User registers herself to the Visited Service Provider

After checking the information, the Visited Service Provider allows or denies the Service to the User.

8.2.1 The Roaming in Detail

When the User moves and wants to use the Service outside the domain of his Home Service Provider she must find a Terminal in the domain of the Visited Service Provider. Then she registers herself asking for the Service (Figure 8.3). This is done through a Mobile Agent.

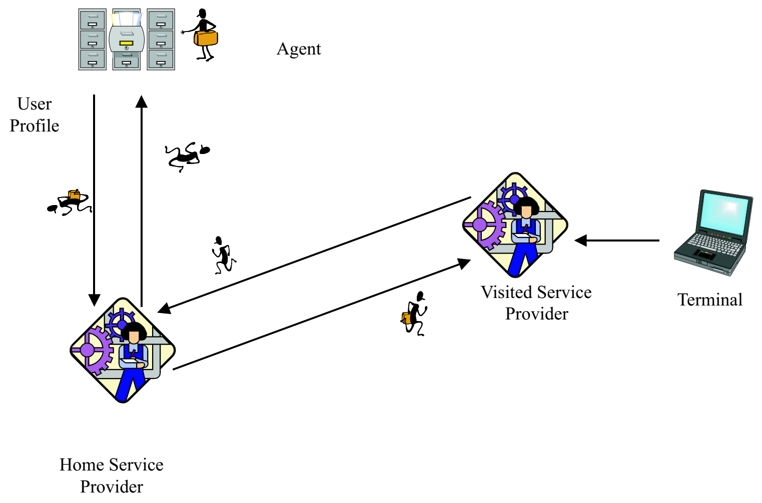


Figure 8.4: The user obtains the service

The Visited Domain receives the request from the Terminal. The request contains also the User's unique *Identifier* so that the Service Provider checks from its Subscription Database and realizes the User does not have a subscription here but that she has roamed. The Service Provider needs to obtain the User's information to decide to allow the Service or not, so the Agent goes to the User's Home Service Provider and gets the information (Figure 8.4).

The scenario becomes more interesting if the Terminal can connect with more than one Service Provider. In this case the User requests not only a service, but also she wants to have the cheapest one (or the best one). In this way the User Agent can add a Quality of Service (QoS) requirement to the Service request.

The role of the Mobile Agents becomes indispensable. Every Service Provider has a Local Agent whose role it is to promote the Service Provider's Services to the visiting Agents. When the User, through the Terminal, requests a Service outside his Home Service Provider, a Mobile

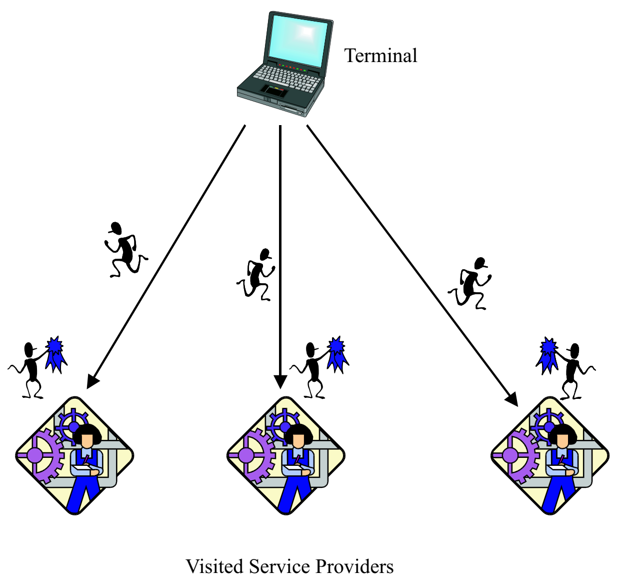


Figure 8.5: Agents negotiate QoS

Agent visits all the local Service Providers and collects the information (promotions) given by the local Service Providers (Figure 8.5).

The Mobile Agent also goes to the Home Service Provider to collect the User Profile and then decides which Service Provider fulfills the User's requirements best. After this phase, the Terminal starts a connection with the chosen Service Provider (Figure 8.6).

8.3 Kiosk Scenario

In this example the Services the User wants to obtain are simple. For example she wants to send a fax document from a Terminal situated in a shop, or she wants to find the cheapest way to travel to a foreigner city from a Kiosk situated in a Travel Agency. This type of Service is not strictly tied to a single Provider, thus the User Information, provided by the User Profile, are directly owned by the User[2]. Therefore the role of the Home Service Provider has no meaning. The Terminal role is also different: Instead of being at the User's side, it is logically attached to the Service Provider. The

[2]Smart Cards could be used for this purpose.

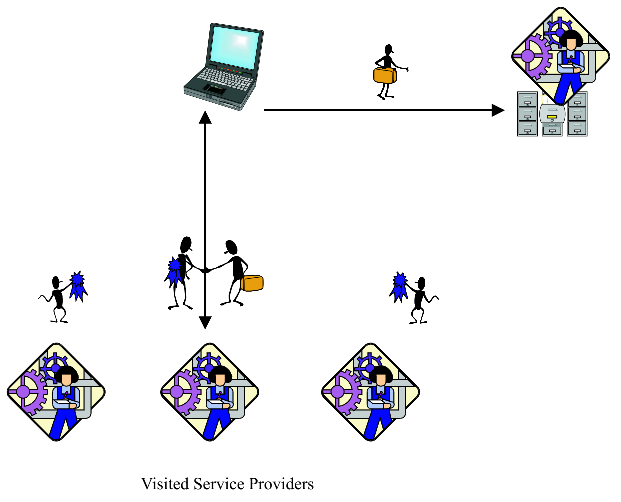


Figure 8.6: After negotiation the Mobile Agent chooses a Service Provider

new scenario is depicted in Figure 8.7.

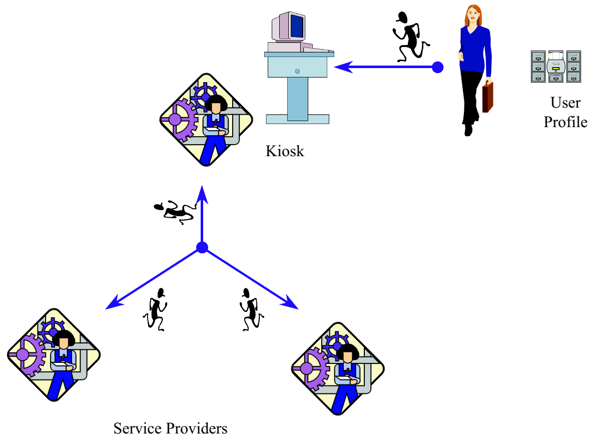


Figure 8.7: Kiosk Scenario

The User connects to a Terminal giving his Identification Code. Since the User Profile is situated in the User domain, the Service Provider immediately gets the information it needs. Then the Service Provider satisfies the User request directly or through other Service Providers.

8.3.1 Booking a Flight though a Kiosk Provider

The User wants to travel to Paris. She goes to a Kiosk Service Provider in the nearest Travel Agency. She puts a Smart Card into a Kiosk Terminal and digits her own Identifier. A Mobile Agent is transferred to the Terminal and an immediate negotiation between these two entities takes place. If the Services is allowed, then another Agent, specialized in Travel Services will get the needed information from the User Profile such as preferred Airline, usual flight class, desired departure and arrival time, and move to the well-known Airlines Providers. Special Agents in those Providers will promote the service and, after the usual negotiation phase, the User will receive the list with the different options and she will choose one. Figure 8.8 depicts the scenario.

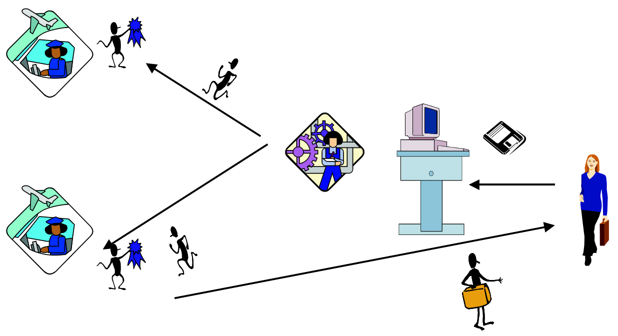


Figure 8.8: Booking a flight

8.3.2 Sending a Fax

Sending a fax through a Kiosk is managed in the same manner: Again, the User inserts a Smart Card into a Terminal connected to the Provider. Also in this case the Smart Card contains the User Profile, but since the Service is a typical anonymous one where the only needed information is a credit card number or a prepaid card, only this information is given to the Provider. The other phases of the Service are similar to the previous ones: the Provider, though a Mobile Agent, finds the cheapest route to send the fax, and then finishes the job.

8.4 Service Invitations

Since in personal mobility the User is not strictly connected with the terminal, new cases involving "Service Invitations" may be interesting. In other words, if the user cannot move the terminal (like a telephone or a fax machine), she would like to have the same services from the Visited Service Provider. As an example, the user would like to have incoming calls to her home fax redirected (diverted) to the visited fax. The user has thus already subscribed to a service at a Service Provider, and the service is active. Since the Service Provider wants to own the user information, the User Profile is tied to it and the Service Provider acts as a Home Service Provider. The user then decides to move to a place with no prior knowledge if the service can be served there or not, so she cannot "a priori" divert the call to a different number. She can just announce to her Home Provider her intention to roam. Once arrived at the destination and when she finds a terminal suitable to receive the service, the User registers itself in the new site to receive the incoming calls. This is done in the same way as the first case: a Mobile Agent scans all the available Connectivity Providers, negotiates the best Quality of Service and decides the best route to reach the Home Service Provider. After this phase, the Agent reaches the Home Service Provider and communicates the new address of the User and the best route to follow (Figure 8.9).

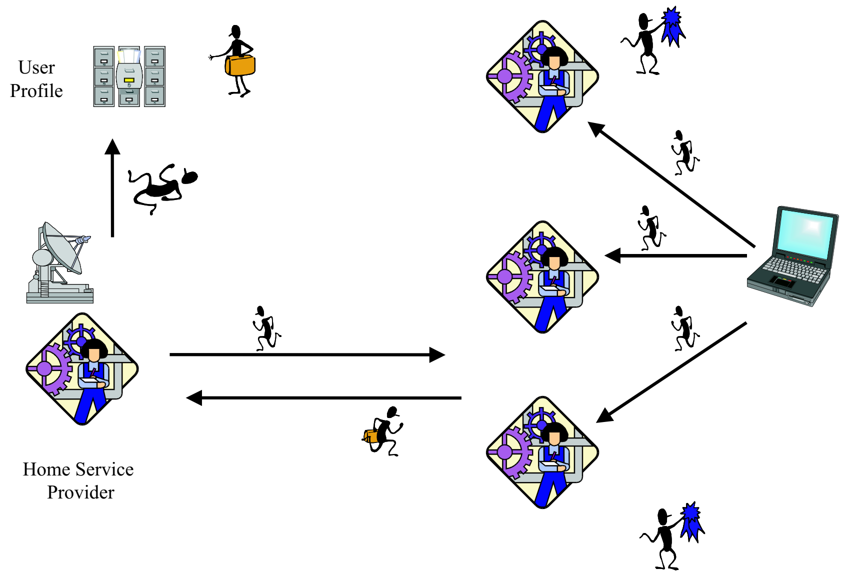


Figure 8.9: Service Invitations

When an incoming call arrives the Home Provide will divert it to the new Address. In this way, if two Users have both roamed to the same Service

Provider's domain, different from the Home one, after the first call they will talk directly, without involving the Home Service Provider any longer and thus minimizing the costs. Another example with the same scenario: A user requests a service that needs a long time to be served. In this case the user could want to receive the answer to another terminal.

8.4.1 Service Invitation Implementation

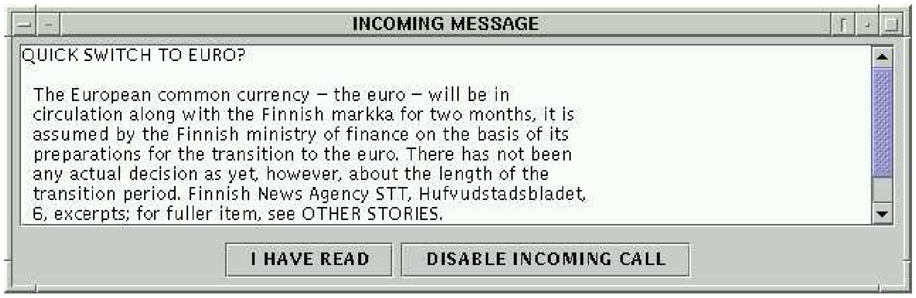
We implemented a simple prototype of this scenario using *Voyager ORB* [97] agent architecture. The scenario is the following:

The User subscribes to a News Service to receive the latest news directly to her video. This service is independent of the user location and the terminal. The Service, when a new news comes, sends it to all the users who have subscribed to the service. The User Agent, through the Agent Platform, takes care to deliver the information to the correct Terminal.

In details The User must register herself to the Service. She has to give her Personal Information Code (PIC) and to specify a host address. In this prototype there is no authentication procedure, and the host specified by the User became her Home Service Retailer.

At this point the User name is inserted in the User Database in the host specified by the User and a User Agent is created.

When the User is willing to receive the News, she starts the Service giving her PIC. The User Agent is awakened and when a news-item arrives it takes care to display it to the User location (Figure 8.10).



QUICK SWITCH TO EURO?

The European common currency – the euro – will be in circulation along with the Finnish markka for two months, it is assumed by the Finnish ministry of finance on the basis of its preparations for the transition to the euro. There has not been any actual decision as yet, however, about the length of the transition period. Finnish News Agency STT, Hufvudstadsbladet, 6, excerpts; for fuller item, see OTHER STORIES.

Figure 8.10: The News Service is active

If the User roams, the Agent will follow her to the new location, thus taking care to obtain the news from the Home Service Provider.

8.4.2 Refinement of the Definitions

As stated in the previous section, the User Profile is not static, but is dynamic or mobile. It is easy to construct new scenarios changing the "position" of the different elements. The main idea is to have few elements and a general architecture valid for a large number of different scenarios. Refining the basic elements described in section 8.1.1 we obtain:

- The User is the entity that requests the services, but from the architectural point of view is not necessary. Also a Service can start another Service (as in the Service Invitation scenario). Only the User Profile is essential.

- The Terminal is just an interface between the User and the Service Provider. This interface can be a real laptop computer, a GSM phone or just a cash dispenser, but the logical function is the same. Sometimes it is situated in the User domain, other times it can be connected to the Service Provider's domain but it is independent from the User.

- The Service Provider is the place where the Service is offered. It is independent from the User. The Home Service Provider is a normal Service Provider with a User Profile.

- The User Profile is the core of the architecture. It contains all the information needed to start a Service. What is needed depends on the scenario: An exhaustive database for a typical telecommunication scenario, a prepaid card for a Kiosk. The main idea is that the User Profile is mobile: it is clear in the fax scenario, since it follows the User's movements, but it could also be just a cash dispenser, but the logical function is the same. Sometimes it is situated in the User domain, other times it can be connected to the Service Provider's domain but it is independent from the User.

- The Mobile Agents are the communication means between the other components. They are not "empty" but they can contain information and intelligence.

8.5 Conclusion

The aim of the examples in this chapter is to introduce the idea to have a special architecture to allow the deployment of Nomadic applications. Even if some definitions are of generic use, the architecture presented in this chapter is tied to a telecommunication scenario. Next chapter introduces a more generic approach to the problem.

Chapter 9

Dynamic Composition of Execution Environment

Any sufficiently advanced technology is indistinguishable from magic.
- Arthur C. Clarke

9.1 Introduction

If the reader has followed the development of the dissertation from the beginning, she will have a clear idea at this point that the solutions presented before are partial and cannot be used in a global context. Improving the transport protocol is a major step in wireless computing but it is not the silver bullet. Adding a specific middleware for adaptation can increase the usability of an application if the communication is prone to sudden disconnections but does not allow the migration of the application to a different device. The application itself can make use of the underlying layers in such a way that makes it impossible to adapt it to any environment different from the one it was designed for. Therefore a different approach is needed. In this chapter we suggest a new paradigm to deploy applications that can run in diverse environments.

9.2 The problem space

The objective we want to achieve with the dynamic composition of the execution environment is an architecture that presents the following characteristics: is Device Independent, is Platform Independent, has a High level of Abstraction, and its adaptation is Transparent to the user.

Normally the applications are designed for a particular environment. This makes it easy for the designer to optimize the application for the characteristics of that environment. It also makes it almost impossible to adapt the same application to a different environment. In fact, Nomadic applications will run on different devices and the *device handover*, or the migration of an application from one device to another with different characteristics, can also happen when the application is in its active state. Our goal is to reach *device independence* so that the application can be executed on a wide variety of devices.

Once active, the application will run on top of an operating system. Our goal is to reach *platform independence*, so that the application can run on top of different operating systems and communication protocols but maintaining the basic application logic. For instance, the application should be able to operate in a Windows or Unix environment and be able to use IIOP or Java RMI as the means of communication.

A high level of *abstraction* is a desired characteristic of any system. This helps the designer to reuse existing solutions or to make new ones available. For this reason we will describe our solution in terms of conceptual modules.

The user should not be forced to manually adapt her application to the new environment when roaming. The ultimate desire of the user is to have the same application anywhere. Since this is not possible due to the different characteristics of the different devices, the adaptation should be *transparent*, so that it should occur without user intervention. On the other hand, the user should be able to monitor the adaptation, and, if desired, to modify it.

9.3 Adaptation Through Dynamic Aggregation

Figure 9.1 depicts how a nomadic application adapts to the existing environment. Once an application is requested to become active, the Personal Agent examines the application logic and the basic modules (both software and hardware) available in the device. It selects the most appropriate hardware modules creating an executing environment. On top of the executed environment, the selected software modules are also aggregated to create an active instance of the application.

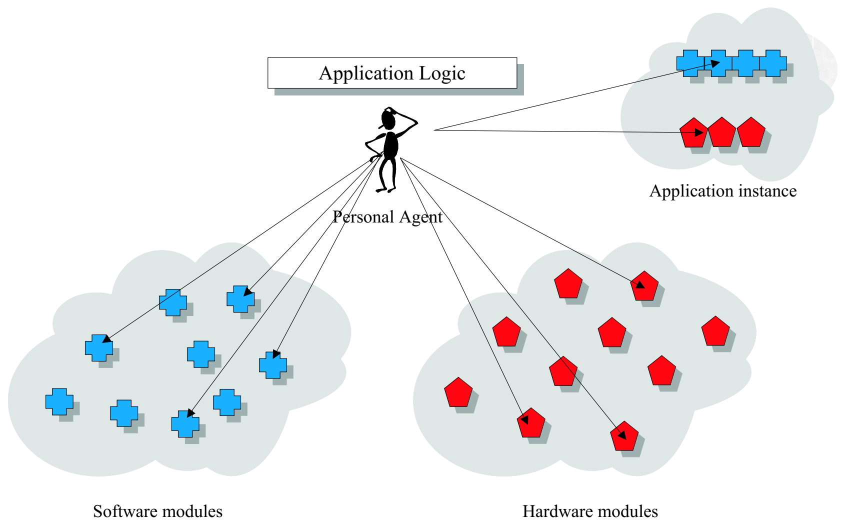


Figure 9.1: Adaptation through dynamic configuration of the execution environment

Adaptation is done by construction: The application instance is built dynamically depending on the characteristics of the device. In the following sections we describe the components of the architecture in detail.

9.3.1 The Basic Modules

One of the components of our architecture is represented by the software and hardware basic modules. The concept of these modules derives from an observation: In a traditional environment, applications often re-implement the same sub-service, like user interfaces or messaging services, instead of reusing already existing instances. In our architecture the services are decomposed into their "smaller" component and the decomposition continues until a bottom level is reached, where further partitioning is not possible without losing the unique characteristic of the service. In this way we create a "community" of services that inter-operate between each other.

As an example of this deconstruction, a web browser application (see Figure 9.2) can be subdivided into smaller services of "communication" and "human interaction". These services can further be decomposed. For example, the "communication" service can be decomposed in a module that

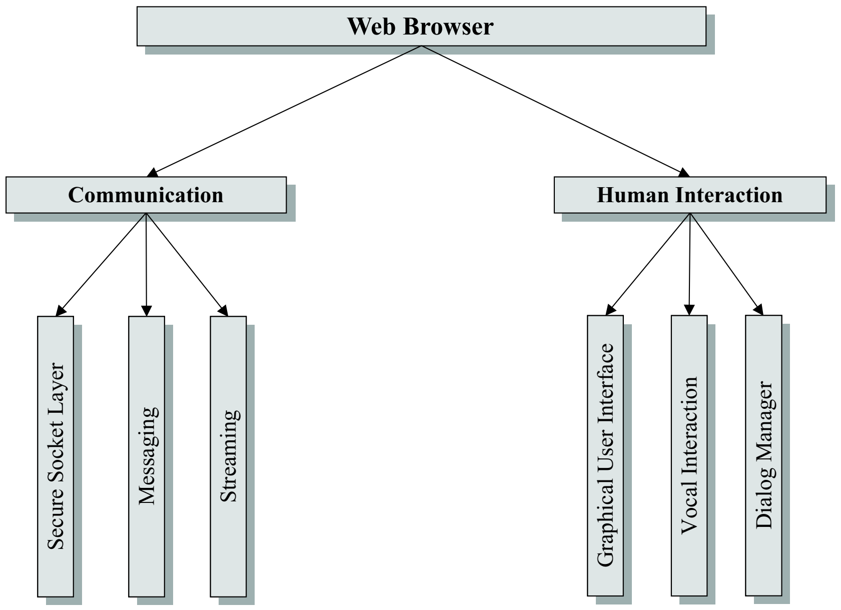


Figure 9.2: Service Deconstruction

implements a secure socket communication, in another module that implements a streaming communication, and so on. Hardware decomposition is done in a similar manner. A desktop computer has several basic modules: the processor that offers computational services, the RAM memory and the hard disks that offer data storage services, the monitor and the speakers that provide output service and the keyboard and the mouse that implement input services. Every basic module implements a basic service and has specific properties. This enables the adaptation by construction: The instance of the application is done by putting together the available basic modules.

9.3.2 Basic module communication and advertisement

In order to be able to aggregate, the basic modules need to communicate with each other. There must be a protocol so that they can offer their services and advertise the proprieties of their services. Furthermore, they need a way to discover which services are offered by other modules and where these other modules are located. The problem space described here is known as ”Service Advertisement and Discovery”. Several solutions have been proposed that can be used. For example, if the community of modules is mostly compound of hardware services, the use of Bluetooth [15] looks appealing. On the other hand, to manage a community of software services we find the use of Jini [6] more interesting if the language environment is Java, or Salutation [87] in promiscuous environments. In any case the protocol, whatever it will be, needs to have clear and open interfaces to avoid the situation, for example, where a community of modules based on Jini is not able to collaborate with a community based on Bluetooth.

9.3.3 Application Logic

Every application can be decomposed in two parts: The Application logic that describes what the application should do, and the state of the application. The application logic needs to be described in a standard way. In our case the application logic should describe the interactions between different modules. It is the task of the Personal Agent to choose an appropriate software module to implement the interaction requested by the application logic.

9.3.4 The Personal Agent

As mentioned before, the Personal Agent has the task to find the most appropriate way to implement the application logic using the available basic modules. The task requires the ability to take sophisticated decisions and to act autonomously. In this dissertation we do not focus on the complex algorithms that the Personal Agent needs to use. We refer the readers to the literature on Intelligent Agents. Instead, we want to focus the attention on the main requirement that our architecture seeks from an agent platform, that is its capability to Interoperate with other platforms.

A further property of the Personal Agent is that it owns the profile of the user. This means it can "a priori" configure the application following the user desire.

9.4 A Sample Application: Incoming News

As an example implementation of our architecture we have the following scenario:

> *A user has a subscription to an information service. When the subject of a news item is of interest to the user, the service*

provider will push the piece of news to the user's device, and the item will be displayed. The user usually receives the news on her desktop at the office but she wants to receive business-related news also when traveling.

9.4.1 Application Logic

The application login of this scenario is quite simple. A sketch is shown in Table 9.1. Basically, the application needs to open a connection with the news server provider, and when a piece of news arrives, to display it on the screen.

Table 9.1: Application Logic

1)	Establish connection to server
2)	Receive description of message
3)	Accept/rejecting incoming message
4)	Display message

9.4.2 Personal Profile

The Personal Agent owns the user profile. Therefore, it knows, for example, that business related news have high priority. It knows also that the user does not like to receive multimedia news if the display in not good enough. The user also expects to be informed about every news, at least about their headline.

9.4.3 Basic Modules

Our example scenario involves two devices. The first one is a desktop computer connected to the network through a fast connection, with a high-resolution color monitor and high computing power. The second device is a smart cellular phone, with wireless connection, low-resolution monitor, limited computing power, and restricted battery life. The basic modules we are interested in in this scenario are described in Table 9.2 and Table 9.3.

9.5 Example of Adaptation

The user enables the application while she is working at the office. The Personal Agent (PA) starts to scan the application logic and inquires from

Table 9.2: desktop Basic Modules

Hardware Modules		Software Modules	
Service	Interface	Service	Interface
Output	ColourDisplay	Compression	Standard
Output	TextDisplay	Messaging	SocketHiBand
Output	StreamingVideoDisplay	Messaging	RMIServer
Output	StereoAudio	...	
Network	fastEthernet		
Processor	HighPower		

Table 9.3: Smart Phone Basic Modules

Hardware Modules		Software Modules	
Service	Interface	Service	Interface
Output	ColourDisplay	Messaging	SocketLowBand
Output	TextDisplay	Messaging	RMIClient
Output	BipAudio	...	
Network	GSMData		
Processor	LowPower		

the community of basic modules for a Messaging service. One service implementing the SocketHiBand interface is available. The related Network service is enabled too. The PA connects to the news service provider. When the description of a new item of news arrives, the PA analyses it and, depending on its characteristics, it requests appropriate Output service. This device has several hardware modules implementing the service, so the application is able to display virtually any kind of news.

The user now decides to move from the office but she desires to keep the news application active in her Smart Phone. The PA takes care of the Device Handover. The application logic is the same but the Basic Modules are different. Therefore the application needs to be modified. The Personal Agent can complete its task in several ways. Here we describe two of them.

1. The PA requests the Messaging service that implements Socket-LoBand and connects to the news server. When the description of the new news item arrives, the PA accepts only the news that can

be transmitted over the wireless connection and shown by the Hardware Basic Modules of the Smart Phone. This implies, for example, discarding all multimedia streams, images and large texts.

2. The PA communicates with another PA situated in the office device. Knowing the user profile, the local PA decides to request of the remote PA the compression of all the images, and, if possible, the creation of a news digest instead of streaming video. The remote PA will carry out these tasks using the desktop device Software Modules. The local PA will then request the Messaging service from the module that implements RMIClient. When the description of a new message arrives, the PA will request its delivery through a Remote Invocation. The modules in the office device will request the news item from the news server, will compress it and send it back as return value of the RMI call.

9.6 Comments

This scenario demonstrates the concept of dynamic composition of the execution environment. The application is constructed dynamically depending on the characteristics of the device in use. The greatest advantage is given by the use of intelligent agents. As in the proposed scenario, the exchange of information between the various Personal Agents can result in innovative solutions. This architecture also opens several issues. One of the most important ones is related to security. The possibility to combine several modules and also to request services from other devices is a powerful enhancement. However, it also introduces several security threats that must be addressed.

9.7 Proof of Concept

The architecture presented in this chapter represents a paradigm shift from the typical application development cycle. Its implementation requires new tools and a new mentality from the application designers. Here we present a simple proof of concept, utilizing existing technologies as much as possible.

9.7.1 A prototype implementation

In this section we present a prototype implementation of the ideas introduced in this chapter. To accelerate the deployment of the prototype we used Java as programming language and RMI (see section 3.3) as remote invocation protocol.

In this example the application (not designed here) needs a means to output a message. Consequently the Personal Agent (PA) has to find the appropriate service between the Basic Modules it knows. The available services depend on the power and the properties of the device where the application is running. This leads to the fact that PA can decide to *modify* the content of the message to meet the request of the service.

Figure 9.3 depicts the environment of this proof of concept: the application requires to display a message to a device. The application logic just requires to invoke the method *output()* without caring what the possibilities of the current device are. It is a task of the PA to return the right service for the device choosing from the ones that have registered to it before.

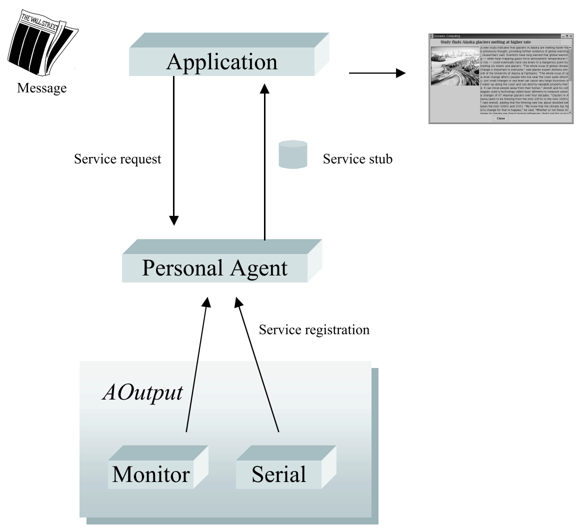


Figure 9.3: The scenario of the proof of concept

9.7.2 Declaring a Basic Module

A service, to become part of the Basic Module Community, has to declare its willingness to join the Community. To do so, it must implement the interface defined in Program 9.1.

Program 9.1 The Java Interface for a generic Basic Module

```
public interface AService {

    public Class getInterface() throws java.rmi.RemoteException;

}
```

This interface contains only one method, `getInterface()`, that returns the interface implemented by the service. This helps the PA to decide what actions have to be taken to fulfill the application needs and, as we will see later, to obtain a reference to the methods that are not defined in this generic interface[1].

9.7.3 Defining a Service

The interface described in Program 9.2 defines a generic Output Service. This is the interface that the client will request when it will need to output a message.

Program 9.2 The Java Interface for a generic Output Service

```
public interface AOutput extends java.rmi.Remote {

    public void output(AMessage msg) throws java.rmi.
            RemoteException;

}
```

This interface defines only one method, `output(AMessage)`. And this is the only method that the client application needs to know. Any service that wants to offer an output service and being part of the Community has to implement both interfaces. But it can also implement other methods, allowing the user to better manage the device.

The class `AMessage` is shown in Program 9.3. Since the message will most probably be sent through the network it has to implement the `Serializable` interface. The class suggests that a message is compounded not only by a string, but it can also contain an image, a headline

[1]Since this proof of concept is implemented in Java, once having obtained a reference, the client is able to find out all its methods and fields by using the Java *reflection* package.

and a priority. The priority can be used to determine which message to display in the case of multiple messages waiting in a queue.

Program 9.3 AMessage class listing (continued on next page)

```
public class AMessage implements Serializable {

    public static final int STANDARD = 0;
    public static final int IMAGE = 1;
    public static final int MULTIMEDIA = 2;

    private String message = null;
    private String headline = null;
    private ImageIcon image=null;
    private int priority=0;
    private int type = STANDARD;

    public AMessage() {}

    public AMessage(String msg){
        this.message = msg;
    }

    public void setMessage(String msg) {
        this.message = msg;
    }

    public String getMessage() {
        return this.message;
    }
    public void setHead(String headline){
        this.headline = headline;
    }

    public String getHead(){
        return this.headline;
    }

    public void setType(int type) {
        this.type = type;
    }

    public int getType(){
        return this.type;
    }
    public void setImage(ImageIcon image){
        this.image=image;
    }

    public void setImageDescription(String description){
        this.image.setDescription(description);
    }

    public String getImageDescription(){
        return this.image.getDescription();
    }

    public ImageIcon getImage() {
        return this.image;
    }

    public void setPriority(int pri) {
        this.priority=pri;
    }

    public int getPriority() {
        return this.priority;
    }

    public boolean isIcon() {
        if (this.image == null)
            return false;
        return true;
    }
}
```

In our prototype there are two services offering an output service: Monitor (whose interface is shown in Program 9.4) and Serial (whose interface is shown in Program 9.5). They both implement the `AOutput` interface, but they also add some different methods. The Serial interface adds a method that sets the color of the text to be displayed (`setColor()`), while the Monitor interface adds a method to define the title of the window displaying the message (`setPanelTitle()`).

Program 9.4 The Monitor Interface

```
public interface IMonitor extends AOutput {


    public void output(AMessage msg) throws java.rmi.
           RemoteException;

    public void setPanelTitle(String title) throws java.rmi.
           RemoteException;


}
```

Program 9.5 The Serial Interface

```
public interface ISerial extends AOutput {

    public void output(AMessage msg) throws java.rmi.
           RemoteException;
    public void setColor(boolean b)throws java.rmi.
           RemoteException;

}
```

9.7.4 Implementing the Service

The class that will offer the service has to implement several interfaces. For instance, the class `AMonitorOutput` (Program 9.6) implements three different Java interfaces:

1. **AService**, thus declaring its willingness to become part of the Community. As such, the method *getInterface()* returns the *IMonitor* interface.

2. **AOutput**, thus declaring that the service it offers implements all the methods describe in that interface. In our case, the method `output(AMessage).`

3. **IMonitor**, thus declaring it also implements all the methods specific to the Monitor interface.

Program 9.6 The implementation class of the Monitor Interface

```
public class AMonitorOutput extends UnicastRemoteObject
        implements  AService, AOutput, IMonitor, java.io.
        Serializable {

    private static String title = "Dynamic Computing";
    private static GUI gui = new GUI(title);

    public AMonitorOutput() throws RemoteException {
        super();
    }


    public Class getInterface() {
        return IMonitor.class;
    }


    public void output(AMessage msg){

        if (msg.getType() == AMessage.MULTIMEDIA)
            JOptionPane.showMessageDialog(null, "Not Standard
                    Message received. Not able to display.", "
                    alert", JOptionPane.ERROR_MESSAGE);
        else {
            gui.output(msg);
            gui.pack();
            gui.show();
        }
    }

    public void setPanelTitle(String title) {

        this.title = title;
    }

    public static void main(String a[]) throws Exception{
      ...

            // Registrering the service with AP
            AMonitorOutput obj = new AMonitorOutput();
            IAP ap = (IAP) Naming.lookup(APlocation);
            ap.register("AOutput","Monitor",obj);
     ...
    }
}
```

The class implements also the interface `Serializable` since this class could be downloaded from the network.

This implementation of the interface `IOutput` utilizes the helper class `GUI` (see Program 9.7), which uses the Java Swing classes to display the message in a window environment. When ready, the service registers itself with the AP using the `register()` method (see also section 9.7.5).

Program 9.7 The helper GUI class listing

```
public class GUI extends JFrame  implements ActionListener {

    public GUI (String  title) {
      super();
      this.setTitle(title);
    }

    public void output(AMessage msg) {

        JPanel headline = new JPanel();
        JPanel text = new JPanel();
        JPanel ok = new JPanel();
        JPanel image = new JPanel();
        ...
        JTextArea  t = new JTextArea(msg.getMessage(),29,30);
        ...
        JLabel img = new JLabel(msg.getImage());
        JButton b1= new JButton("Close");
        b1.setActionCommand("read");
        b1.addActionListener(this);
        headline.add(h);
        ...
    }

    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();
        if (command.equals("read"))
            hide();
    }
}
```

The `AOutput` service is also implemented by another class, `ASerialOutput` (see Prog. 9.8). This service is aimed to display the message to a device that has no ability to display windows.

Program 9.8 The listing of the ASerialOutput class

```
public class ASerialOutput extends UnicastRemoteObject implements
        AOutput, AService, ISerial, java.io.Serializable {

    private boolean color = false;

    public ASerialOutput() throws RemoteException{
        super();
    }

    public Class getInterface() {
        return ISerial.class;
    }

    public void output(AMessage msg) {
        int type;

        System.out.println(msg.getHead()+"\n");
        type=msg.getType();
        if (type != AMessage.STANDARD){
            System.out.print("WARNING: Not a standard message. ")
                    ;
        switch (type) {
        case AMessage.MULTIMEDIA:
            System.out.println("This device cannot display it.\n"
                    );
            break;
        case AMessage.IMAGE:
            System.out.println("Cannot display image: "+msg.
                    getImageDescription()+"\n");
            break;
        }
        }
        if(msg.getPriority() != 0)
            System.out.println("HIGH PRIORITY:");
        System.out.println(msg.getMessage());
    }


    public void setColor(boolean b) {
        this.color=b;
    }

    public static void main(String a[]) throws Exception{
      ...

      // Registering with AP
            ASerialOutput obj = new ASerialOutput();
            IAP ap = (IAP) Naming.lookup(APlocation);
            ap.register("AOutput","Serial",obj);
      ...
    }
}
```

9.7.5 The Personal Agent

The Personal Agent has a central role in this prototype. It receives the registration from the services by the `register()` method and it has to deliver the services that the applications require through the `getService()` method. Thus a generic Personal Agent has to implement the interface described in Program 9.9.

Program 9.9 The Personal Agent interface

```
public interface IAP extends Remote {

    Remote getService(String service) throws RemoteException;
    void register(String service, String inter, Remote stub)
           throws RemoteException;

}
```

Program 9.10 The listing of the ServiceCouple class

```
public class ServiceCouple {
    private String ident;
    private Remote stub;

    public ServiceCouple(String id, Remote st){
        this.ident=id;
        this.stub=st;
    }

    public Remote getStub(){
        return this.stub;
    }

    public String getId(){
        return this.ident;
    }
}
```

Program 9.11 shows a prototype implementation of a Personal Agent. The `register()` method is implemented on line 13. The first parameter declares the "family" of the services it implements (i.e. AOutput). The second parameter gives information about the capabilities of the specific implementation (such as "Monitor" or "Serial") while the last parameter is the RMI stub.

The AP groups all the stubs of the same family in a vector (line 18), and then puts the vector in a hash table (line 19). It also uses the `Service Couple` helper class (Program 9.10) to store the stubs.

A client application will request a service through the method `getService()` implemented on line 28. The client will only request the "family" of the service (in our case `AOutput`), and it is the task of the AP to give the one that fits the device best. In our prototype the logic is quite simple:

1. If there are no services satisfying the request, raise an exception (line 33)
2. If there is only a service satisfying the request, return that (lines 38-40)
3. If there are both services, return the Monitor one, that is, the one with the best QoS (lines 43-47)

Program 9.11 The listing of the AP class

```
public class AP extends UnicastRemoteObject implements IAP {

 // The refernces of the services are kept is a hash table
   private static Hashtable services_hash = new Hashtable();

   ...

   public AP() throws RemoteException {
       super();
   }


   public void register(String service, String stub_int, Remote
          stub) throws RemoteException{

     ServiceCouple sc = new ServiceCouple(stub_int,stub);

       if(!services_hash.containsKey(service)){
         v.add(sc);
         services_hash.put(service,v);
         }
       else {
         Vector ser = (Vector)services_hash.get(service);
           if(!ser.contains(sc))
              ser.add(sc);
       }
   }

   public Remote getService(String service) throws
          RemoteException{

       ServiceCouple sc = null;

       if(!(services_hash.containsKey(service)))
           throw new RemoteException("Service not registrered");
       else {
           Vector ser = (Vector)services_hash.get(service);

           //If there is only one service, we return that
           if (ser.size()==1) {
               sc=(ServiceCouple)ser.get(0);
               return sc.getStub();
           }
           // Otherwise we return Monitor (Hacked!)
           sc = (ServiceCouple)ser.get(0);
           if (sc.getId().equals("Monitor"))
               return sc.getStub();
           sc = (ServiceCouple)ser.get(1);
           return sc.getStub();
       }
   }
}
```

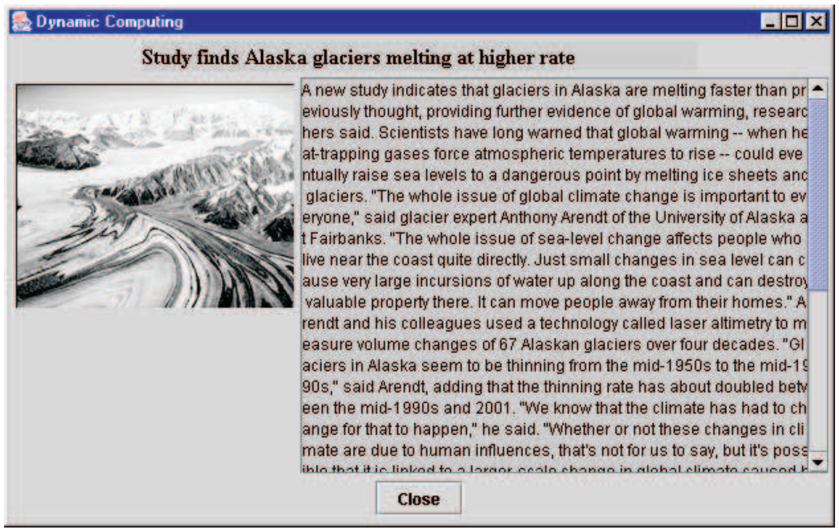
9.7.6 The Client

An example of the application client is shown on Program 9.12. It creates a message, connects to the AP and requests a service implementing the AOutput service. Then it uses the received service to display the message.

Program 9.12 The listing of the AClient class

```
public class AClient {

    public static void main(String a[]) throws Exception{

      ...


            AMessage msg = new AMessage();
            msg.setMessage(...);
            msg.setHead("Study finds Alaska glaciers melting at
                   higher rate");
            msg.setImage(new ImageIcon("glacier.jpg"));
            msg.setImageDescription("Glaciers in Alaska.");
            msg.setType(AMessage.IMAGE);
            IAP ap = (IAP) Naming.lookup(APlocation);
            AOutput out = (AOutput) ap.getService("AOutput");
            out.output(msg);
     ...

    }
}
```

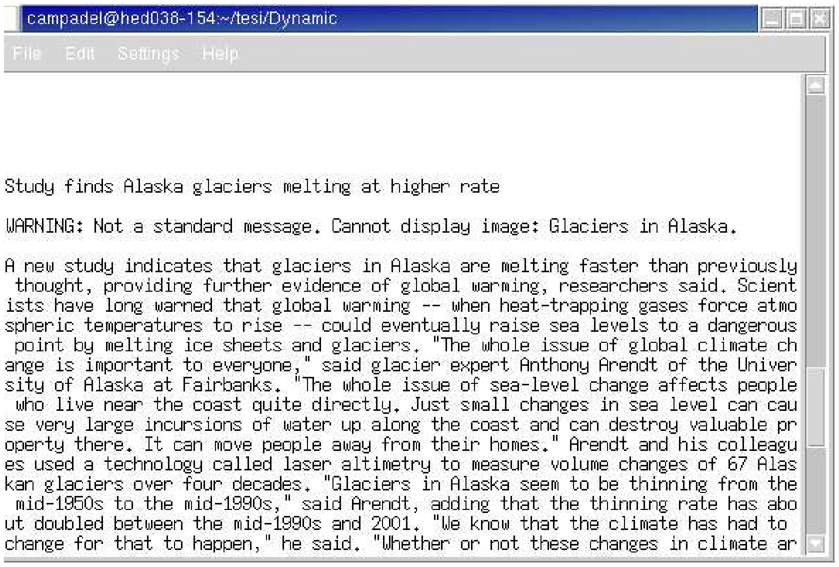
The quality of the display depends upon which service implementation the PA returned. Figure 9.4 shows the result on a Windows host with the Monitor class, while Figure 9.5 shows the same message displayed to a Linux console using the Serial class. The fact that the adaptation to the device capabilities (and operating system) has happened dynamically at runtime and without client intervention should be stressed.



Study finds Alaska glaciers melting at higher rate

```
A new study indicates that glaciers in Alaska are melting faster than pr
eviously thought, providing further evidence of global warming, researc
hers said. Scientists have long warned that global warming -- when he
at-trapping gases force atmospheric temperatures to rise -- could eve
ntually raise sea levels to a dangerous point by melting ice sheets and
 glaciers. "The whole issue of global climate change is important to ev
eryone," said glacier expert Anthony Arendt of the University of Alaska a
t Fairbanks. "The whole issue of sea-level change affects people who
live near the coast quite directly. Just small changes in sea level can c
ause very large incursions of water up along the coast and can destroy
 valuable property there. It can move people away from their homes." A
rendt and his colleagues used a technology called laser altimetry to m
easure volume changes of 67 Alaskan glaciers over four decades. "Gl
aciers in Alaska seem to be thinning from the mid-1950s to the mid-19
90s," said Arendt, adding that the thinning rate has about doubled betw
een the mid-1990s and 2001. "We know that the climate has had to ch
ange for that to happen," he said. "Whether or not these changes in cli
mate are due to human influences, that's not for us to say, but it's poss
ible that it is linked to a larger scale change in global climate caused b
```

Figure 9.4: The message displayed with the Monitor class



```
Study finds Alaska glaciers melting at higher rate

WARNING: Not a standard message. Cannot display image: Glaciers in Alaska.

A new study indicates that glaciers in Alaska are melting faster than previously
 thought, providing further evidence of global warming, researchers said. Scient
ists have long warned that global warming -- when heat-trapping gases force atmo
spheric temperatures to rise -- could eventually raise sea levels to a dangerous
 point by melting ice sheets and glaciers. "The whole issue of global climate ch
ange is important to everyone," said glacier expert Anthony Arendt of the Univer
sity of Alaska at Fairbanks. "The whole issue of sea-level change affects people
 who live near the coast quite directly. Just small changes in sea level can cau
se very large incursions of water up along the coast and can destroy valuable pr
operty there. It can move people away from their homes." Arendt and his colleagu
es used a technology called laser altimetry to measure volume changes of 67 Alas
kan glaciers over four decades. "Glaciers in Alaska seem to be thinning from the
 mid-1950s to the mid-1990s," said Arendt, adding that the thinning rate has abo
ut doubled between the mid-1990s and 2001. "We know that the climate has had to
change for that to happen," he said. "Whether or not these changes in climate ar
```

Figure 9.5: The message displayed with the Serial class

Part V

Conclusions

Chapter 10

Conclusions

Life can only be understood backwards; but it must be lived forwards.
- Johann Wolfgang von Goethe

10.1 The Journey of this Dissertation

In this dissertation we described the evolution of the concept of Nomadic Computing. This journey followed a pattern that is quite common in the Information Technology world. Once new ideas become concrete, there is a tendency to apply them in all possible fields. This step creates further ideas but also new challenges. Sometimes the challenges become too difficult, and the evolution paths of the new ideas reach their ends. At other times these new ideas coherently evolve in a new paradigm and bring innovation in different fields, even not correlated with each other before. But most of the time they just stand in the middle, being able to evolve and innovate only in one direction.

The improvement in the wireless infrastructure during the last decade has pushed many researchers to apply this technology to computer networks, trying to create a new paradigm able to melt together two of the most successful branches of technology: Telecommunication and Computer Science. This new paradigm has many names. We used *Nomadic Computing*. But this has been a marriage more of interest than love. In Chapter 2 we described the differences and the challenges they had to face, and then in Chapter 4 the solutions they were offered to. But we think this is not enough. In Chapter 5 we believe to have demonstrated that putting together those two technologies trying to cut off their differences is not enough. We designed and implemented a working prototype that gave surprisingly good results. That happened because we had in our mind both partners when we put them together.

At the same time this marriage produced new ideas that related them even closer, as we described in Chapter 3. But it is our belief that we need more. So, in the last part of the dissertation we described what we believe to be the best compromise: Both partners *have to change themselves* in order to create new features. It is a new *forma mentis*, where the two technologies become indistinguishable as they simply offer services. It is the relationship between the different services that changes depending upon the location and time, not the single service. This, in our opinion, makes the two technologies able to *adapt* to each other.

10.2 The World Outside

The paradigm described by the author in the last part of this dissertation is original, but lately some of those ideas have been proposed independently in other contexts and in other forms. For example, the role of the intelligent agents behind the concept of *Semantic Web* [13, 91] is very similar to the role of the Personal Agent described in Section 9.3.4. Also, FIPA [36] has spent efforts to standardize the presence of intelligent agents in a nomadic environment as, for example, the specification on Nomadic Application Support [38].

As an example on the application side, the World Wireless Reference Forum [105] pushes the concept of dynamic adaptation as an important point in its vision for the future of telecommunications [104].

10.3 Final Remarks

Unfortunately, even if new ideas are constantly appearing to conjugate the richness of modern applications with the restricted proprieties of mobile devices, overall there has not been such an organic view as depicted through this dissertation. So far the marriage between the two technologies, as argued above, is still in its early phase.

We hope that this dissertation will add stimulus to the development of the paradigm envisaged in its last chapters. We believe there is a need to make a distinction between the application logic, or what an application is supposed to do, and its implementation, or how it will reach its scopes. Nomadic Computing would take full advantage of this, since more emphasis would be put on fulfilling the users' desires and needs. And the evolution of Nomadic Computing is the ultimate goal of this dissertation.

References

[1] ACM 1998 Workshop on Java for High-Performance Network Computing. Available from the World Wide Web: `<http://www.cs.ucsb.edu/conferences/java98/program.html>`.

[2] T. Alanko, M. Kojo, H. Laamanen, M. Liljeberg, M. Moilanen, and K. Raatikainen. Measured Performance of Data Transmission over Cellular Telephone Networks. *Computer Communications Review*, 24(5):24–44, October 1994.

[3] Alice Web Site. Available from the World Wide Web: `<http://www.dsg.cs.tcd.ie/research/alice/>`.

[4] M. Allman. On the generation and use of TCP acknowledgments. In *ACM Computer Communication Review*, volume 28(5), October 1998.

[5] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

[6] K. Arnold, O. Sullivan, R. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification*. Addison-Wesley, 1999.

[7] A. Aziz and W. Diffie. Privacy and authentication for wireless local area networks. *IEEE Personal Communications*, 1:25–31, 1994.

[8] B. R. Badrinath, Arup Acharya, and Tomasz Imielinski. Impact of mobility on distributed computations. *ACM Operating Systems Review*, 27(2):15–20, April 1993.

[9] R. Bagrodia, W. Chu, L. Kleinrock, and G. Popek. Vision, issues, and architecture for nomadic computing. *IEEE Personal Communications*, pages 14–27, December 1995.

[10] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving TCP/IP Performance over Wireless Networks. In *Proc. of the First ACM International Conf. on Mobile Computing and Networking (MobiCom '95)*, pages 2–11, Berkeley, California, USA, November 1995.

[11] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. Challenges: An application model for pervasive computing. In *Proceedings of The sixth Annual International Conference on Mobile Computing and Networking*, pages 266–274, August 2000.

[12] F. Berg, S. Diwan, J. Villacis, J. Balasubramanian, E. Akman, and D. Gannon. Java RMI Performance and Object Model Interoperability: Experiments with Java/HPC++. In *Proc. of the ACM 1998 Workshop on Java for High-Performance Network Computing*, Stanford, Palo Alto, Calif., February 1998.

[13] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[14] V. Bharghavan and V. Gupta. A framework for application adaptation in mobile computing environments. In *Proceedings of COMPSAC 1997*, August 1997.

[15] The Bluetooth specification. Available from the World Wide Web: `<http://www.bluetooth.com/dev/specifications.asp>`.

[16] E. Brewer, R. Katz, Y. Chawathe, S. Gribble, T. Hodes, G. Nguyen, M. Stemm, T. Hender-Son, E. Amir, H. Balakrishnan, A. Fox, V. Padmanabhan, and S. Seshan. A network architecture for heterogeneous mobile computing. *IEEE Personal Commun.*, pages 8–24, 1998.

[17] R. Cáceres and L. Iftode. The Effects of Mobility on Reliable Transport Protocols. In *Proc. IEEE 14th International Conference on Distributed Computer Systems*, pages 12–20, Poznan, Poland, June 1994. IEEE Computer Society Press.

[18] R. Cáceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, 13(5):850–857, June 1995.

[19] S. Campadello. Dynamic composition of execution environment for adaptive nomadic applications. In Samuel Pierre and Roch Glitho, editors, *Mobile Agent for Telecommunication Applications. Proceedings of the Third International Workshop, MATA 2001*, Lecture Notes in Computer Science, pages 73–80, Montreal, Canada, August 2001. Spinger.

[20] S. Campadello, H. Helin, O. Koskimies, P. Misikangas, M. Mäkelä, and K. Raatikainen. Using Mobile and Intelligent Agents to Support Nomadic Users. In *6th International Conference on Intelligence in Networks (ICIN2000)*, pages 199–204, Bordeaux, France, January 2000.

[21] S. Campadello, H. Helin, O. Koskimies, and K. Raatikainen. Performance enhancing proxies for Java2 RMI over slow wireless links. In *Proceedings of the Second International Conference on the Practical Application on Java*, pages 76–89, Manchester, UK, April 2000.

[22] S. Campadello, H. Helin, O. Koskimies, and K. Raatikainen. Wireless Java RMI. In *Proceedings of The 4th International Enterprise Distributed Object Computing Conference*, pages 114–123, Makuhari, JAPAN, September 2000. IEEE Computer Society. ISBN 0-7695-0865-0.

[23] S. Campadello and K. Raatikainen. Agent in personal mobility. In *Proceeding of the First International Workshop on Mobile Agents for Telecommunication Application, MATA 99*, pages 359–374, Ottawa, Canada, October 1999. World Scientific.

[24] S. Casner and V. Jacobson. Compressing IP/UDP/RTP headers for low-speed serial links. RFC 2508, February 1999.

[25] The Component Object Model specification. Microsoft Corporation. Available from World Wide Web: `http://www.microsoft.com/com/resources/comdocs.asp`.

[26] IEEE Personal Communications. Special Issue on the European Path Towards UMTS. Vol. 2, 1, February 1995.

[27] NOKIA Corporation. Mobile IPv6. In *Inside MITA*, pages 65–76. IT Press, 2001.

[28] DCOM technical overview. Microsoft Corporation, November 1996. Available from the World Wide Web: `<http://msdn.microsoft.com/library/backgrnd/html/\\/msdn\_dcomtec.htm>`.

[29] M. Degermark, B. Nordgren, and S. Pink. IP header compression. RFC 2507, February 1999.

[30] T. Dierks and C. Allen. The TLS protocol. Technical Report RFC 2246, IETF, 1999. Available from the World Wide Web: `<http://www.ietf.org/rfc/rfc2246.txt>`.

[31] I. Dittrich, P. Holzner, and M. Krumpe. Implementation of the GSM-Data-Services into the mobile radio system. *MCR Mobile Radio Conference, Nice France Nov. 1991*, pages 73–83, 1993. GSM Data Services.

[32] Object Management Group. Telecom DTF. Wireless access and terminal mobility in corba draft adopted specification. OMG document dtc/01-05-01, May 2001.

[33] Endeavour Expedition: Charting the Fluid Information Utility . Available from the World Wide Web: `<http://endeavour.cs.berkeley.edu/>`.

[34] M. Engan, S. Casner, and C. Bormann. IP header compression over PPP. RFC 2509, February 1999.

[35] M. Esler, J. Hightower, T. Anderson, and G. Borriello. Next century challenges: Data-centric networking for invisible computing: The portolano project at the university of washington. In *Proceedings of The fifth Annual International Conference on Mobile Computing and Networking*, 1999.

[36] *Foundation for Intelligent Physical Agents (FIPA) Internet Homepage.* Available from the World Wide Web: <http://www.fipa.org>.

[37] Foundation for Intelligent Physical Agent. FIPA ACL message representation in bit-efficient specification, October 2000. Specification number XC00069.

[38] Foundation for Intelligent Physical Agent. FIPA agent message transport envelope representation in bit-efficient specification, November 2000. Specification number XC00088.

[39] Foundation for Intelligent Physical Agent. FIPA nomadic application support specification, November 2000. Specification number XC00014.

[40] G. Forman and J. Zahorjan. *The Challenges of Mobile Computing*. Computer Science Department, University of Washington, U.S., 1993.

[41] S. D. Gribble, M. Welsh, J. R. von Behren, E. A. Brewer, D. E. Culler, N. B., S. E. Czerwinski, R. Gummadi, J. R. Hill, A. D. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Y. Zhao. The Ninja architecture for robust internet-scale systems and services. *Computer Networks*, 35(4):473–497, 2001.

[42] GSM Technical Specification, GSM 02.34. High Speed Circuit Switched Data (HSCSD), Stage 1, July 1997. Version 5.2.0.

[43] GSM Technical Specification, GSM 02.60. GPRS Service Description, Stage 1, 1998. Version 6.1.0.

[44] GSM Technical Specification, GSM 03.34. High Speed Circuit Switched Data (HSCSD), Stage 2, May 1999. Version 5.2.0.

[45] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. Technical Report RFC 2608, IETF, 1999. Available from the World Wide Web: <http://www.ietf.org/rfc/rfc2608.txt>.

[46] M. Haahr, R. Cunningham, and V. Cahill. Supporting CORBA Applications in a Mobile Environment. In *Proc. of the ACM MobiCom '99 Conference*, pages 36–47, Seattle, Wash., August 1999.

[47] R. Hagen. Security requirements and their realization in mobile networks. In *XIV International Switching Symposium*, pages 127–131, Yokohama, 1992. IEICE. GSM Security;TELE Library.

[48] B. Hausel and D. B. Lindquist. WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. In *Proc. of the Second ACM International Conf. on Mobile Computing and Networking (MobiCom '96)*, pages 108–116, Rye, New York, USA, November 1996.

[49] H. Helin. *Supporting Nomadic Agent-based Applications in FIPA Agent Architecture*. PhD thesis, University of Helsinki, 2003. ISBN 952-10-0882-2.

[50] H. Helin and S. Campadello. Providing messaging interoperability in FIPA communication architecture. In K. Zielinski, K. Geihs, and A. Laurentowski, editors, *New Development in Distributed Applications and Interoperable System. Proceedings og the Third IFIP TC6/WG6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS)*, pages 121–126, Krakow, Poland, September 2001. Kluwer Academic Publishers.

[51] S. Hirano, Y. Yasu, and H. Igarashi. Performance Evaluation of Popular Distributed Object Technologies for Java. In *Proc. of the ACM 1998 Workshop on Java for High-Performance Network Computing*, Stanford, Palo Alto, Calif., February 1998.

[52] P. Honeyman, L. Huston, J. Rees, and D. Bachmann. *The LITTLE WORK Project*. Third Workshop on Workstation Operating Systems. IEEE Computer Society Press, Key Biscayne, Florida, U.S., 1992.

[53] T. Imielinski and B.R. Badrinath. *Mobile Wireless Computing: Solutions and Challenges in Data Management*. The State Univeristy of New Jersey RUTGERS, 1993. General Interest Data Management.

[54] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM '88 Symposium on Communications Architectures and Protocols*, pages 314–329, Stanford, California, August 1988.

[55] V. Jacobson. Compressing TCP/IP Headers for Low-Speed Serial Links. Request for Comments 1144, Network Information Center, February 1990.

[56] D. Johnson and C. Perkins. Mobile support in IPv6. Technical report, IETF, 2001. Available from World Wide Web: `http://www.ietf.org/internet-drafts/draft-ietf-mobileip-ipv6-15.txt`.

[57] A. D. Joseph, J. A. Tauber, and M. F. Kaashoek. Mobile Computing with the Rover Toolkit. *IEEE Transactions on Computers (Special Issue on Mobile Computing)*, 46(3), March 1997.

[58] L. Kleinrock. Nomadicity: Anytime, anywhere in a disconnected world. *Mobile Networks and Applications.*, January 1997.

[59] L. Kleinrock. Nomadic computing and smart spaces. *IEEE Internet Computing*, pages 52–53, Jan-Feb 2000.

[60] M. Kojo, K. Raatikainen, and T. Alanko. Connecting Mobile Workstations to the Internet over a Digital Cellular Telephone Network. In *Proc. of the MOBIDATA Workshop*. Rutgers University, NJ, November 1994. Updated version in Mobile Computing, Kluwer, 1996, pages 253–270.

[61] M. Kojo, K. Raatikainen, M.Liljeberg, J.Kiiskinen, and T.Alanko. An Efficient Transport Service for Slow Wireless Telephone Links. *IEEE Journal on Selected Areas in Communications*, 15(7):1337–1348, September 1997.

[62] H. Laamanen, H. Helin, and S. Campadello. Software agent technology in nomadic computing: FIPA Nomadic Appication Support. In *XIII International Symposium on Services and Local accesS (ISSLS2000)*, Stockholm, Sweden, June 2000.

[63] B. Lampson et al. Authentication in distributed systems. *ACM Trans. On Computer Systems 10,4 Nov 92*, pages 265–311, 1993. Distributed Systems Authentication.

[64] J. Landay. User interface issues in mobile computing. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*. IEEE, October 1993.

[65] M. Liljeberg, H. Helin, M. Kojo, and K. Raatikainen. Enhanced Service for World-Wide Web in Mobile WAN Environment. Technical Report C-1996-28, Department of Computer Science, University of Helsinki, 1996.

[66] M. Liljeberg, K. Raatikainen, M. Evans, S. Furnell, N. Maumon, E. Veltkamp, B. Wind, and S. Trigila. Using CORBA to Support Terminal Mobility. In *Proc. of TINA'97 Conference*, pages 56–67. IEEE Computer Society Press, 1998.

[67] J. Maassen, R. van Nieuwpoort, R. Veldema, H. E. Bal, and A Plaat. An Efficient Implementation of Java's Remote Method Invocation. In *Proc. 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 173–182, Atlanta, GA, May 1999.

[68] P. Maniatis, M. Roussopoulos, E. Swierk, K. Lai, G. Appenzeller, X. Zhao, and M. Baker. The mobile people architecture. In *ACM Mobile Computing and Communications Review (MC2R)*, July 1999.

[69] Sun Microsystems. Java Remote Method Invocation – Distributed Computing for Java. White Paper, 1998.

[70] P. Misikangas, M. Mäkelä, and K. Raatikainen. Predicting QoS for Nomadic Applications Using Intelligent Agents. In *Proc. of the IMPACT99*, Seattle (WA), December 1999.

[71] The MIT project Oxygen. Available from the World Wide Web: <http://oxygen.lcs.mit.edu/>.

[72] Monads home page. Available from the World Wide Web: <http://www.cs.helsinki.fi/research/monads/>.

[73] R. Monson-Haefel. *Enterprise JavaBeans*. O'Reilly Publisher, 3 edition, 2001.

[74] G. Montenegro, S. Dawkins, M. Kojo, V. Magret, and N. Vaidya. Long Thin Networks. RFC 2757, January 2000.

[75] Mosquitonet: The mobile computing group at stanford university. Available from the World Wide Web: <http://mosquitonet.stanford.edu/>.

[76] Mowgli Home Page. Available from the World Wide Web: <http://www.cs.Helsinki.FI/research/mowgli/>.

[77] B. J. Nelson. Remote procedure call. Technical Report CSL-81-9, Xeros Palo Alto Research Center, 1981.

[78] C. Nester, M. Philippsen, and B. Haumacher. A More Efficient RMI for Java. In *Proc. of ACM 1999 Java Grande Conference*, pages 152–157, San Francisco, Calif., June 1999.

[79] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K. Walker. Agile application-aware adaptation for mobility. *ACM SIGOPS Oper. Syst. Rev.*, 5(31):276–287, 1997.

[80] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 1995. Available from the World Wide Web: `<http://www.omg.org/corba2/>`.

[81] Object Management Group. Telecom DTF. RFP on Wireless Access and Terminal Mobility in CORBA. OMG document telecom/99-05-05, May 1999.

[82] Object Management Group Internet Homepage. Available from the World Wide Web: `<http:www.omg.org/>`.

[83] C. Perkins. IP mobility support. Technical Report RFC 2002, IETF, 1996. Available from the World Wide Web: `<http://www.cis.ohio-state.edu/htbin/rfc/rfc2002.html>`.

[84] The portolano project home page. Available from the World Wide Web: `<http://portolano.cs.washington.edu/>`.

[85] M. Rahnema. Overview of the GSM System and Protocol Architecture. *IEEE Communication Magazine*, 31(4):92–100, April 1993.

[86] Rover Web Site. Available from the World Wide Web: `<http://www.pdos.lcs.mit.edu/rover/>`.

[87] The Salutation specification. Available from the World Wide Web: `<http://www.salutation.org>`.

[88] M. Satyanarayanan. Fundamental challenges in mobile computing. *Fifteenth ACM Symposium on Principles of Distributed Computing*, May 1996.

[89] M. Satyanarayanan. Pervasive computing: Vision and challanges. *IEEE Personal Communications*, pages 10–17, August 2001.

[90] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4), April 1990.

[91] The Semantic Web Community Portal. Available from World Wide Web: `<http://www.semanticweb.org/>`.

[92] A. Shacham, R. Monsour, R. Pereira, and M. Thomas. IP payload compression protocol (ipcomp). RFC 2393, December 1998.

[93] R. Srinivasan. RPC: Remote Procedure Call protocol specification version 2. RFC 1831, August 1995.

[94] Third Generation Partnership Project Web Site. Available from the World Wide Web: `<http://www.3gpp.org/>`.

[95] The universal plug and play forum home page. Available from the World Wide Web: `<http://www.upnp.org/>`.

[96] R. van Nieuwpoort, J. Maassen, H. E. Bal, T. Kielmann, and R. Veldema. Wide-area parallel computing in Java. In *Proc. of ACM 1999 Java Grande Conference*, pages 8–14, San Francisco, Calif., June 1999.

[97] *ObjectSpace Voyager ORB Internet Homepage*. Available from the World Wide Web: `<http://www.objectspace.com/products/voyager/>`.

[98] Xhtml 1.0: The extensible hypertext markup language. Available from World Wide Web: `http:www.w3.org/`.

[99] Helen J. Wang, Bhaskaran Raman, Chen nee Chuah, Rahul Biswas, Ramakrishna Gummadi, Barbara Hohlt, Xia Hong, Emre Kiciman, Zhuoqing Mao, Jimmy S. Shih, Lakshminarayanan Subramanian, Ben Y. Zhao, Anthony D. Joseph, and Randy H. Katz. ICEBERG: An Internet-core network architecture for integrated communications. *IEEE Personal Communications*, 2000. Special Issue on IP-based Mobile Telecommunication Networks.

[100] WAP Forum. Wireless Application Environment Overview, version 1.2, November 1999. Available from World Wide Web: `http://www.wapforum.org/what/technical.htm`.

[101] Wap home page. Available from the World Wide Web: `<http://www.wapforum.org/>`.

[102] M. Weiser. Some computer science issues in ubiquitous computing. *Comm. of The ACM 36,7 Jul 93*, pages 74–84, 1992. General Interest Application.

[103] J. E. White. A high-level framework for network-based resource sharing. *Proc. National Computer Conference*, June 1976.

[104] WWRF. Book of vision 2001, 2001. Available from World Wide Web: `<http://www.wireless-world-research.org/Bookofvisions/Bov.html>`.

[105] Wireless World Research Forum. Available from World Wide Web: `<http://www.wireless-world-research.org/>`.

A-1995-1	P. Myllymäki: Mapping Bayesian networks to stochastic neural networks: a foundation for hybrid Bayesian-neural systems. 93 pp. (Ph.D. thesis).
A-1996-1	R. Kaivola: Equivalences, preorders and compositional verification for linear time temporal logic and concurrent systems. 185 pp. (Ph.D. thesis).
A-1996-2	T. Elomaa: Tools and techniques for decision tree learning. 140 pp. (Ph.D. thesis).
A-1996-3	J. Tarhio & M. Tienari (eds.): Computer Science at the University of Helsinki 1996. 89 pp.
A-1996-4	H. Ahonen: Generating grammars for structured documents using grammatical inference methods. 107 pp. (Ph.D. thesis).
A-1996-5	H. Toivonen: Discovery of frequent patterns in large data collections. 116 pp. (Ph.D. thesis).
A-1997-1	H. Tirri: Plausible prediction by Bayesian inference. 158 pp. (Ph.D. thesis).
A-1997-2	G. Lindén: Structured document transformations. 122 pp. (Ph.D. thesis).
A-1997-3	M. Nykänen: Querying string databases with modal logic. 150 pp. (Ph.D. thesis).
A-1997-4	E. Sutinen, J. Tarhio, S.-P. Lahtinen, A.-P. Tuovinen, E. Rautama & V. Meisalo: Eliot – an algorithm animation environment. 49 pp.
A-1998-1	G. Lindén & M. Tienari (eds.): Computer Science at the University of Helsinki 1998. 112 pp.
A-1998-2	L. Kutvonen: Trading services in open distributed environments. 231 + 6 pp. (Ph.D. thesis).
A-1998-3	E. Sutinen: Approximate pattern matching with the q-gram family. 116 pp. (Ph.D. thesis).
A-1999-1	M. Klemettinen: A knowledge discovery methodology for telecommunication network alarm databases. 137 pp. (Ph.D. thesis).
A-1999-2	J. Puustjärvi: Transactional workflows. 104 pp. (Ph.D. thesis).
A-1999-3	G. Lindén & E. Ukkonen (eds.): Department of Computer Science: annual report 1998. 55 pp.
A-1999-4	J. Kärkkäinen: Repetition-based text indexes. 106 pp. (Ph.D. thesis).
A-2000-1	P. Moen: Attribute, event sequence, and event type similarity notions for data mining. 190+9 pp. (Ph.D. thesis).
A-2000-2	B. Heikkinen: Generalization of document structures and document assembly. 179 pp. (Ph.D. thesis).
A-2000-3	P. Kähkipuro: Performance modeling framework for CORBA based distributed systems. 151+15 pp. (Ph.D. thesis).
A-2000-4	K. Lemström: String matching techniques for music retrieval. 56+56 pp. (Ph.D.Thesis).
A-2000-5	T. Karvi: Partially defined Lotos specifications and their refinement relations. 157 pp. (Ph.D.Thesis).
A-2001-1	J. Rousu: Efficient range partitioning in classification learning. 68+74 pp. (Ph.D. thesis)
A-2001-2	M. Salmenkivi: Computational methods for intensity models. 145 pp. (Ph.D. thesis)
A-2001-3	K. Fredriksson: Rotation invariant template matching. 138 pp. (Ph.D. thesis)
A-2002-1	A.-P. Tuovinen: Object-oriented engineering of visual languages. 185 pp. (Ph.D. thesis)
A-2002-2	V. Ollikainen: Simulation techniques for disease gene localization in isolated populations. 149+5 pp. (Ph.D. thesis)
A-2002-3	J. Vilo: Pattern Discovery from Biosequences. 149 pp. (Ph.D. thesis)
A-2003-1	J. Lindström: Optimistic Concurrency Control Methods for Real-Time Database Systems. 111 pp. (Ph.D. thesis)
A-2003-2	H. Helin: Supporting Nomadic Agent-based Applications in the FIPA Agent Architecture. 200+17 pp. (Ph.D. thesis)