Implementations Of User Mobility Support for UPC in JAVA/CORBA Environment

by

Thong Tri Huynh

B.A.Sc., University of British Columbia,1993

B.Sc., University of British Columbia, 1996

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia

August 1999

In presenting this thesis/essay in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying for this thesis for scholarly purposes may be granted by the Head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Aug 30, 99
Date

Computer Science
The University of British Columbia
2366 Main mall
Vancovuer, BC
Canada V6T 1Z4

Abstract

This thesis addresses the specification and implementation of the Universal Personal Computing (UPC) environment, developed at UBC for support of user mobility. In the UPC computing environment, each user has a unique global identification with which the user can access network services, computing resources, personal applications, data files, and environmental configuration through *any* terminal, stationary or mobile, wired or wireless networking, *anywhere* on the Internet.

In this thesis, we produced the specification of the UPC's Registration and Service Negotiation Protocol (RSNP) using the Specification and Description Language (SDL) and simulated the RSNP using a SDL supported tool. As an initial attempt to verify the concept, we implemented the UPC protocol and a graphical user interface using C and Tcl/Tk as the programming language. CORBA and Java offers an alternative approach in modelling the UPC concept that is flexible and portable to any computing platform. A UPC prototype is implemented using CORBA as the architecture framework and Java as the programming language.

Contents

Abstract ii

Contents iii

List of Figures vi

Acknowledgements vii

1 Introduction 1

1.0.1 Motivation 1

1.0.2 Thesis Objectives and Contributions 4

1.0.3 Thesis Outline 6

2 Background 7

2.1 Mobile Host Protocol for IP (Mobile-IP) 7

2.2 Specification Description Language (SDL) 10

2.3 Universal Personal Computing (UPC) 11

2.3.1 User Identification 12

2.3.2 User Profile 12

2.3.3 Terminal Identification 13

2.3.4 Terminal Profile 13
2.3.5 UPC Network Architecture 13
2.3.6 Registration and Authentication 15
2.4 Introduction to CORBA 16
2.4.1 Interface Definition Language (IDL) 18
2.4.2 Object Request Broker (ORB) 19
2.4.3 CORBA Object Services (CORBAservices) 20
2.4.4 The CORBA Common Facilities (CORBAfacilities) 22
2.5 CORBA-based UPC 22
2.5.1 Facility object 23
2.5.2 Personal Computing Environment (PCE) object 23
2.5.3 Terminal Profile object 24
2.5.4 User Agent (UA) object 24
2.5.5 Terminal Agent (TA) object: 25
2.5.6 Initial Agent (IA): 25

3 UPC's Registration and Service Negotiation Protocol (RSNP) 27

3.1 Overview 27
3.2 UPC Protocol Architecture Modification 29
3.3 SDL Specification of RSNP 31
3.3.1 Pseudo Mobile-IP Block 35
3.3.2 Mobile User Block 36
3.3.3 User Foreign Agent Block 38
3.3.4 User Home Agent Block 41
3.4 Prototype Implementation of UPC 41
3.4.1 Assumptions and Requirements: 45

3.4.2 Registration and Service Negotiation Procedure 46
3.4.3 Re-registration after Expiration of Connection Lifetime . . . 47
3.4.4 Re-registration after Hand-off 48
3.4.5 Datagram Format used in Registration and Service Negotiation Protocol 48

4 CORBA-based UPC Prototype Implementation 52
4.1 CORBA-based UPC Prototype Implementation 52
4.1.1 Personal Computing Environment (PCE) 54
4.1.2 User Home Agent (UHA) 54
4.1.3 Terminal Profile 55
4.1.4 Terminal Agent (TA) 58
4.1.5 Initial Agent 60
4.1.6 Testing and Results 63

5 Conclusion and Future Works 66
5.1 Conclusion 66
5.2 Future Works 68

Bibliography 70

Appendix A 75

List of Figures

2.1 CORBA Object Architecture [23] 17

2.2 IDL Language Bindings Provide Interoperability [23] 18

2.3 ORB structure [23] 20

2.4 Collaboration of agents in the CORBA-based UPC framework [30] . 26

3.1 System Diagram of UPC's Registration and Service Negotiation Protocol 32

3.2 Block Diagram of a Pseudo Mobile-IP 36

3.3 Block Diagram of a Mobile User 37

3.4 Mobile User Agent's Extended Finite State Machine 39

3.5 Block Diagram of a User Foreign Agent 40

3.6 User Foreign Agent's Extended Finite State Machine 42

3.7 Block Diagram of a User Home Agent 43

3.8 User Home Agent's Extended Finite State Machine 44

4.1 Main screen of the Service Manager utility 57

4.2 Login screen of the UPC 61

4.3 UPC Application Manager screen 62

Acknowledgements

I would like to take this opportunity to thank Dr. Son Vuong, my supervisor, who has guided, supported and encouraged me throughout the Master program and especially this thesis. I would also like to thank my second reader, Dr. George Tsiknis, for the time and valuable suggestions to this thesis. Many thanks to Maria Toeroe, Jinsong Zhu, and Kangming Liu for all the lengthy meetings and discussions on the UPC's issue. It has been a pleasure to work with all of you. Finally, I thank my parents and my sisters for their love and support on every stage of my education.

THONG TRI HUYNH

The University of British Columbia

August 1999

Chapter 1

Introduction

1.0.1 Motivation

In recent years, the term "Internet" has become a household word. The cost of connecting to the Internet has become affordable for everyone, and most people work in an environment where their desktop computer is connected to the Internet. Internet applications such as Email, World Wide Web, Internet chat (e.g. IRC, Microsoft Chat), Internet video conferencing (e.g. CuSeeMe, SDR, NetMeeting), and Internet radio (e.g. RealPlayer) are widely used on computers at home and at work.

Meanwhile, portable computers have become more compact, yet more powerful, with increased processing and storage capabilities. They are available in increased diversity, ranging from laptop, to notebook and palmtop computers. In addition, the communication capabilities of these portable computers are impressive as high-speed modems, PCMCIA modem, and gigabit satellite access, and so on become more common [1].

At the same time, wireless technology has grown significantly, especially

wireless technologies for LANs. In the market place we can find inexpensive high-speed wireless LAN products, in addition to low-speed wide area wireless network and medium-speed metropolitan area wireless network services. This availability of wireless network services enables mobile users with portable computers to remain connected to the Internet anywhere at any time.

Nowadays, people are constantly on the move. Many people are moving between offices, homes, automobiles, conference rooms, and so forth. While they are on the move, they may want to be able to perform computing and/or communication. As they move around, they may have access to enormous computing facilities such as advanced workstations, desktop computers, compact PDAs (Personal Digital Assitant), and convenient locally available services, such as faxes and printers. However, they often find themselves de-coupled from their familiar "home base" computing environment when using these foreign computing facilities. It would be better if the user could have the same personal computing environment, regardless of the computer's platform, the computer's location, or the user's location.

The combination of increasing use of portable computers, the rapid advancement of wireless communication technologies, and the fact that most users are mobile, requires a special computing environment that supports mobile computing. Mobile computing refers to an area of technology that aims at providing access to computing resources, independent of motion, location, computing platform, communication device, and communication bandwidth. The independence refers to the concept of a computing environment that automatically adjusts to the processing, communications and access available at any moment. Ideally, a user can travel from place to place and still use the computing facilities they desire in their new location or even while they are on the move.

Currently, a user can obtain mobile computing by installing a cellular telephone interface into their portable computer and running a protocol such as SLIP or PPP over a dialup service. This approach is simple and cheap; however, there is a severe limitation in that the user can only access Internet resources within a single administrative domain. This limitation is due to the Internet's inability to support mobility, neither user nor terminal mobility. In regards to terminal mobility, the current version of the Internet protocol is built on an implicit assumption that the point at which a computer is attached to the Internet is fixed, and its IP address uniquely identifies this point of attachment. The IP address is used by the Internet's routing protocol to deliver the IP packets to the intended receiver. When a computer moves to a different network, its network services are lost because its IP address does not reflect its new point of attachment to the Internet, and the routing protocol is unable to correctly route the packets to it. In this situation, the computer must be reconfigured with a new IP address associated with the new network. Doing so causes the lost of already-established transport layer connections (for example, ftp or telnet sessions). Similar limitations apply to mobile users, where each user is registered to one administrative domain and granted a user login ID that allows the user to access only the computing resources and network services in the registered administrative domain. As the user moves to another administrative domain, they are not able to access any network services.

Several approaches have been proposed and implemented to enhancing the Internet protocol to support mobility. A standard protocol (Mobile Host Protocol for Internet Protocol) that supports terminal (or host) mobility has recently been defined by the Internet Engineering Task Force (IETF) as in RFC2002 [25]. This protocol is often referred to as mobile-IP. Mobile-IP allows a terminal to roam

freely on the Internet while still maintaining one permanent IP address. However, supporting only terminal mobility is not sufficient to provide a seamless mobile computing environment. Even with mobile-IP support, a mobile user still can not login to any available terminal, stationary or mobile, in a foreign administrative domain. As a result, this introduces a need for an additional level of mobility on top of terminal mobility: user (or personal) mobility. Personal mobility is different from terminal mobility. Terminal mobility refers to the ability of a portable terminal to access services from any location while in motion, and the capability of the network to locate and identify the mobile terminal as it moves [10]. On the other hand, personal mobility refers to the ability of the end user to send and receive calls, and to access subscribed services on any terminal in any location, and the ability of the network to identify the end user as they move [10].

1.0.2 Thesis Objectives and Contributions

Supporting both terminal and personal mobility is the fundamental requirement of mobile computing; however, very little work has been done to support personal mobility on the Internet. A new computing paradigm that supports both terminal mobility and personal mobility over the Internet, called Universal Personal Computing (UPC), has been developed [20,35] jointly in the Department of Computer Science and the Department of Electrical and Computer Engineering of the University of British Columbia. This research constitutes a major activity of Internetworking Wireless Data Networks for Mobile Computing (IWIN), a project jointly funded by the NSERC of Canada and Motorola Canada. UPC is a computing environment which enables a mobile user to access computing resources, network services, personal applications, data files, and environmental configurations through any ter-

minal, stationary or mobile, anywhere on the Internet [21]. In this environment, terminal mobility and personal mobility are managed separately. Terminal mobility is supported using mobile-IP with some enhancements in TCP-connection management; whereas personal mobility is managed via an agent-based architecture.

The core of UPC's mobility management is the Registration and Service Negotiation protocol (RSNP). In this thesis, we use the Specification and Description Language (SDL) to produce a clear and precise specification of RSNP. We also simulate RSNP using a SDL supported tool to verify for RNSP completeness and correctness. In addition, the RSNP simulation can be used to simulate alternative solutions for any problems that come up. After having a concrete understanding of RSNP, we implement a UPC prototype using C and Tcl/Tk as the programming language.

In the proposed UPC architecture [21, 35] there are distributed components that can execute transparently to the host location, host platform and component implementation. The distributed nature of UPC's components makes the implementation of UPC very complex and even impossible. Fortunately, several technologies that support platform independent distributed object architecture, such as CORBA and DCOM, are available. Combining of concept of the UPC and CORBA technology leads to a CORBA-based UPC architecture as proposed in [30, 35]. In this new architecture, the UPC components are modeled as distributed objects with CORBA as the object bus that facilitates the interaction, integration, and distribution of these objects. In this thesis, we implement a prototype of a CORBA-based UPC system using Java as the programming language, Visigenic's VisiBroker for Java as the CORBA framework, and Sun Microsystem's Solaris 2.x as the computing platform.

1.0.3 Thesis Outline

The rest of the thesis is organized as follows:

- Chapter 2 provides the necessary background material including mobile-IP, overview of the Specification and Description Language (SDL), the concept of UPC, introduction to CORBA, and CORBA-based UPC.
- Chapter 3 presents UPC's RSNP formal specification using SDL, and a prototype of UPC's RSNP using C and Tcl/Tk.
- Chapter 4 presents the prototype the of CORBA-based UPC.
- Chapter 5 concludes the thesis and discusses possible future works.

Chapter 2

Background

2.1 Mobile Host Protocol for IP (Mobile-IP)

The current version of Internet Protocol (IP v4) has been built on the implicit assumption that the point of attachment of a computer to the Internet is fixed and uniquely identified by its IP address. IP packets sent to a computer are delivered based on the location information in the IP packet's headers, i.e. the destination IP address. When a mobile computer moves to a new network, the IP address that is configured in the computer does not reflect the new point of attachment. Thus, the IP packets that are destined to this particular computer are incorrectly routed by the routing protocol. To overcome this routing problem, the mobile computer must be reconfigured with a new IP address that reflects the new point of attachment in the visiting network. Unfortunately, although the suggested method above may solve the routing problem, it presents two new problems. The first problem is the loss of any established transport and higher-layer connection during handoff when changing the IP address. The second problem rests on the burden

of informing potential correspondents of the new IP address. Therefore, we need a new mechanism to support the mobility of a computer (often referred as host) over the Internet.

Mobile Host Protocol for IP (Mobile-IP) is an enhancement to IP that allows a mobile computer with a permanent IP address to communicate data transparently to application software, and independent of its current point of attachment to the Internet. The Mobile-IP is standardized by the Internet Engineering Task Force (IETF) as defined in the RFC2002 [25]. The standard Mobile-IP supports host mobility independent of the communication medium in use. It works as effectively in a wireless environment as it does in a wired internetworking environment. The Mobile-IP architecture consists of a set of special entities called the Mobile Host [MH], the Home Agent [HA], and the Foreign Agent [FA].

Mobile Host (MH): A MH is defined as a host that changes its point of attachment to the Internet from one network to another. Each MH belongs to a unique home network and is assigned a permanent IP address. When a MH is away from the home network, it is associated with a temporary IP address that is often referred as a care-of IP address. The care-of IP address identifies the MH's current point of attachment to the Internet. A care-of IP address could be either the IP address of the FA that the MH is registered to, or a local IP address that is assigned to the MH at the visiting network.

Home Agent (HA): HA is the router on a MH's home network that maintains the current location information (i.e. care-of IP address) for all mobile hosts under its administration. The HA is responsible for intercepting IP packets destined for a MH and tunnelling these IP packets to the MH via their care-of

address when the MH is away from the home network.

Foreign Agent (FA): FA is a router on a MH's visited network. The FA is responsible for de-tunneling and delivering IP packets sent from the MH's HA to the registered MH. A FA also serves as a default router for IP packets sent by a registered MH.

The HA and FA advertise their presence in the network using an Agent Advertising message. The Agent Advertising message is a router advertisement message with a special Extension attachment. When a MH connects to a network, the MH receives the Agent Advertisement message and identifies whether its current point of attachment is on its home network or on a foreign network. If the MH is on its home network, there is no need for mobility services. However, if the MH has just returned to its home network from a foreign network, de-registration with the HA is required. As a MH detects it is on a foreign network, it obtains its care-of IP address on the foreign network and uses a special registration protocol to keep its HA informed about its current location (i.e. care-of IP address).

When a MH is away from the home network, IP packets destined to it are always first sent to its home network, and then encapsulated by the HA and resent to the MH's current FA. Encapsulation refers to the process of enclosing the original IP packet as data inside another IP packet with a new IP header. The routing information in the original IP packet, source and destination address, remain unchanged. The outer IP packet header specifies the MH's care-of IP address as the destination address and the MH's HA IP address as the source address. The encapsulated IP packet is then sent to the care-of IP address via a conventional IP routing mechanism. Without IP encapsulation, IP packets meant for the MH

are always routed to the MH's HA by intermediate routers since all these packets have MH's permanent IP address as the destination address in their headers. Upon receiving an encapsulated IP packet, the FA extracts the inner IP packet from the encapsulated IP packet and delivers the extracted packet to the appropriate visiting MH on its local network.

Conversely, when the MH sends IP packets, they are directly routed by the FA to their destination using conventional IP routing mechanisms. Passing these IP packets through the MH's HA is not necessary.

2.2 Specification Description Language (SDL)

The Specification Description Language (SDL), developed and standardized by Comite Consultatif International Telegraphique et Telephonique (CCITT), is a formal specification language. The SDL can be applied to specify a variety of systems from telecommunication systems, data communication systems, real time systems and interactive systems, to distributed systems.

The SDL focuses on the reactive behaviour of a system. It is concerned with how external stimuli and responses are related at system interfaces, instead of the internal and physical construction of a system. The SDL can be used in both textual and graphical representations. The graphical SDL allows the user to specify a system using a set of diagrams that cover different levels of detail from a broad overview down to the detailed design level. The textual SDL looks similar to a programming language. Fragments of textual SDL can be embedded in diagrams where text is more suitable or the graphical symbol does not exist (e.g. signal declarations, data declarations).

In SDL, the top level of description is a system. A SDL system represents

significant properties of a real life application system. Anything that is not part of the SDL system is part of the environment. At the top level, a first overview of the system is shown without getting into unnecessary details. In the SDL system we may have a single SDL block or a set of SDL blocks which are the building blocks of a system. These SDL blocks interact with each other and the environment via channels. The SDL blocks build the system structure and allow decomposing a system into a hierarchy of levels in which each level represents a different level of abstraction.

A SDL block may contain other SDL blocks or SDL processes. A Process is the lowest level description of a system; it can not be further decomposed. We can describe SDL processes as extended Finite State Machines (FSM). Each SDL process works autonomously and concurrently with other SDL processes. SDL processes communicate with each other using discrete messages called signals. Each signal has a name, sender identity, and possible additional data. The FSM uses the name of the signal to make a transition from one state to another state.

2.3 Universal Personal Computing (UPC)

Universal Personal Computing (UPC) is a computing paradigm that supports location independent network computing over the Internet. UPC is a similar concept to Universal Personal Telecommunication (UPT),which has been defined by the ITU-T, yet UPC caters to network computing over the Internet, rather than telecommunications. In the UPC environment, each user has a unique global identification. Using this identification, the UPC network enables users to access network services, computing resources, personal applications, data files, and environmental configuration through any terminal, stationary or mobile, wired or wireless networking, anywhere

on the Internet.

2.3.1 User Identification

In UPC, each user is assigned a Logical User Identifier (LUI) that is globally unique and independent of the user's current location. As proposed in Li and Leung [21], a simple LUI could be a user's email address. It is composed of the login user ID of a user at their home network concatenated with the user's home domain address. Using the email address as LUI has several advantages. First of all, email addresses are familiar to any Internet user; thus, it is easy to remember and reference. Secondly, the existing DNS could be used to resolve the user home agent's IP address from the user's home domain address portion in the LUI. "This obviates the need for global user-names server, and is compatible with the current Internet architecture" [21].

2.3.2 User Profile

Another component of UPC is the User Profile. Each mobile user has a service profile kept in a user database at their home network. The service profile contains authentication information, such as the user login ID and the corresponding password. The service profile also defines a set of computing resources, network services, and personal applications that a mobile user is entitled to while at home or on the road. For each personal application, computing resource, or network service, there is a set of user preferences, such as environmental requirements, configuration, default settings, and so on. Modifying or updating one's service profile can only be performed by an authorized network administrator, or by the user themselves. If necessary, the mobile user is allowed to remotely update or modify their service

profile while away from the home network.

2.3.3 Terminal Identification

In the UPC environment, each terminal is identified by a Logical Terminal Identifier (LTI). In the current IETF standard of Mobile-IP, each mobile terminal (MT) is associated with two IP addresses, in particular the MT's permanent IP address and the MT's care-of IP address. The MT's permanent IP address is used in identifying the MT at the network level, and the MT's care-of IP address is used in discovering the MT's current location. IP packets addressed to a MT are redirected to the MT by tunneling to the MT's care-of IP address. Since we want to achieve backward compatibility with the current Internet protocol suite, we selected the MT's permanent IP address as the MT's LTI, and the MT's care-of IP address provides the MT's current location. We can then use the same tunneling mechanism as in mobile-IP to redirect packets to the MT's current location.

2.3.4 Terminal Profile

The terminal profile is used to re-create the computing environment specified by a user. Each mobile terminal has a terminal profile kept in a terminal database at the mobile terminal's home network. The terminal profile defines the capabilities on the terminal, such as the resident operating system, the file format, the graphical user interface, the display mode and monitor resolution.

2.3.5 UPC Network Architecture

UPC network is an agent-based architecture that consists of three functional entities: a Terminal Home Agent (THA), a User Home Agent (UHA), and a Foreign Agent

(FA).

User Home Agent (UHA): Each administrative domain has a User Home Agent. The UHA maintains a database of all the users registered in this administrative domain as well as their associated user profiles. Along with the user profile, the UHA also keeps a record of all users' current location information. This location information is a binding information between the mobile user (i.e. LUI), the terminal (i.e. LTI) that the mobile user is currently using, and the current location of the terminal (i.e. care-of IP address). Whenever the user moves or changes their association with a terminal, the location information is updated.

Terminal Home Agent (THA): In addition to the UHA, each administrative domain has a Terminal Home Agent (THA). The THA maintains a database of all the mobile terminals that the network is configured to serve, as well as their associated terminal information, such as the terminal identity, the terminal profile, the terminal authentication key, and the current location of the terminal (i.e. care-of IP address).

Foreign Agent (FA): If the administrative domain serves mobile users, it has a Foreign Agent. The FA enables the user and mobile terminal to be temporarily use the network so that registration and authentication can proceed. After the user and terminal successfully register, the FA's main tasks are packet redirection and possibly caching pertinent parts of the user's service profile. As part of network management, the FA maintains a list of all the mobile users and all the mobile terminals that are currently visiting the network.

2.3.6 Registration and Authentication

In the UPC environment, user mobility is handled independently from terminal mobility; thus, registration is separated into two registration procedures, terminal registration for the mobile terminal and user registration for the mobile user. Terminal registration must be performed before user registration so that the terminal is recognized by the foreign network and ready for user registration. Both terminal registration and user registration are mandatory for any mobile user to access a mobile terminal in either their home network or a foreign network.

Terminal Registration

In our terminal registration procedure, we adopt the MT's registration procedure as specified in Mobile-IP. The sequence of events for terminal registration is as follows:

1. The MT registers with the FA in the visited network.
2. The FA contacts the MT's THA to authenticate the MT and inform the THA of the MT's new care-of IP address.
3. Upon successful registration and authentication, the terminal profile of the MT is transferred to the FA.

User Registration

User Registration is similar to terminal registration, and follows the steps below:

1. The MU sends a registration request to the FA in the visiting network.
2. The FA contacts the MU's UHA to authenticate the MU. It also provides to the MU's UHA, the IP address of the MT's THA that the MU is currently on.

3. The MU's UHA retrieves the MT's terminal profile from the MT's THA and then evaluates a set of suitable services for the MU.

4. Upon successful registration and authentication, a set of services available for the MU at this particular MT is returned to FA, and then relayed to the MU.

2.4 Introduction to CORBA

The Common Object Request Broker Architecture (CORBA), proposed by the Object Management Group (OMG), is a new software technology that combines object-oriented technology and distributed client-server computing to provide an industrial standard distributed object architecture. CORBA enables applications to interact with one another without knowing where the corresponding application resides, how the corresponding application is implemented, or what operating system the corresponding application is executed on.

CORBA presents a strong, universal, powerful distributed object-oriented framework that makes software design easier. An application that is developed using the CORBA architecture is highly portable and interoperable across a heterogeneous distributed environment. CORBA not only provides a framework for developing new applications, but it also provides a framework for integrating existing client/server applications.

This section provides a tour of major components of CORBA such as the Interface Definition Language (IDL), the Object Request Broker (ORB), the CORBA Object Services (CORBAservices), and the CORBA Common Facilities (CORBfacilities). Figure 2.1 presents CORBA's Object Architecture .

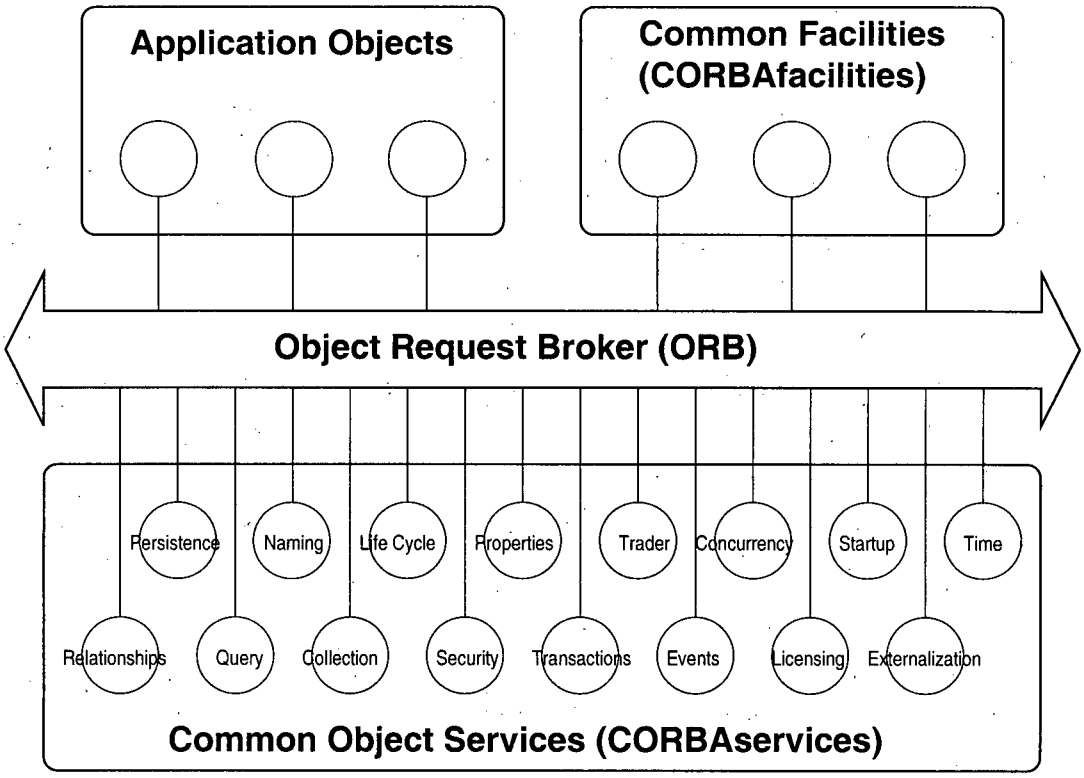

Figure 2.1: CORBA Object Architecture [23]

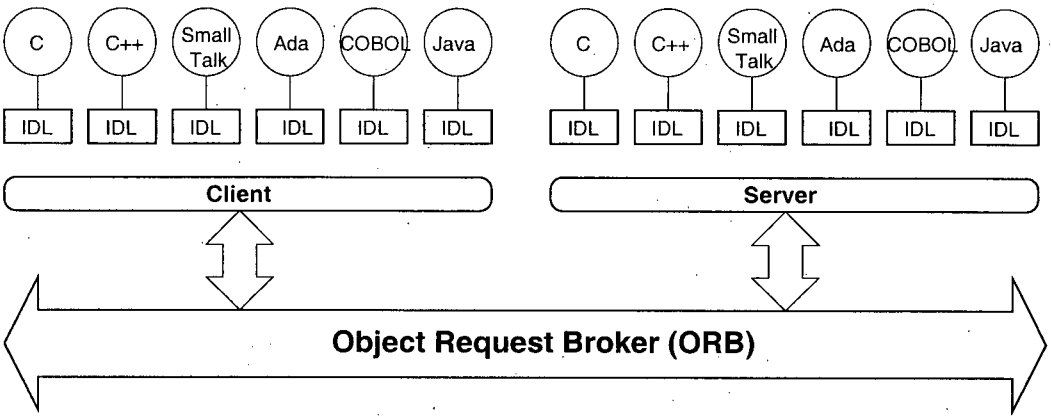

Figure 2.2: IDL Language Bindings Provide Interoperability [23]

2.4.1 Interface Definition Language (IDL)

In the CORBA architecture, all objects are specified using the Interface Definition Language (IDL). IDL is the key for interoperability in CORBA. The IDL is a language that is used to describe the external behaviour of an object without providing any underlying implementation details about an object. The IDL makes a strong separation between the specification of an object and the implementation of that object. Objects that use the interfaces published from another object have no idea how the object is implemented. Thus, IDL enables an object written in one language to communicate with another object written in an unknown language. Figure 2.2 presents the IDL interoperability concept.

The IDL grammar uses the same lexical rules as C++, introduced with several new keywords to support distributed-computing concepts. An object's attributes, base classes that it inherits from, exceptions it raises, methods it supports, can all be specified using IDL. Objects specified using IDL can be mapped into a particular programming language or an object system. The IDL specifications are compiled into header files, stub, and skeleton programs which are used by a "bro-

ker" program (i.e. Object Request Broker) which allows the objects to communicate with one another.

2.4.2 Object Request Broker (ORB)

The ORB is the central component of CORBA. In the OSI network model, the ORB sits between the data and application layers. The ORB is an object bus that provides the mechanisms by which objects can make requests to other objects and receive responses from other objects. The caller object is often referred to as a client and the corresponding object is called a server. At runtime, when a client object invokes a method in a server object, ORB is responsible for locating the server object implementation that can implement the method, and then invoking the method and returning the results to the client object. The client object does not have to be aware of the server object's location, implementation, nor its operating system.

The ORB supports both static and dynamic method invocations. The static method invocation interfaces are defined at compile time and presented to the client as stub code. The dynamic method invocation interfaces are dynamically discovered at run time using the CORBA Interface Repository. An Interface Repository is an on-line database that contains real-time information describing all the interfaces that an object supports along with its parameters. The structure of the ORB is shown in figure 2.3.

An ORB can run in standalone mode or can be interconnected to other ORBs in the universe. All the ORBs are interoperable via the Internet Inter-ORB Protocol (IIOP). "The IIOP is basically a TCP/IP with some CORBA-defined message exchanges that serve as a common backbone protocol" [23].

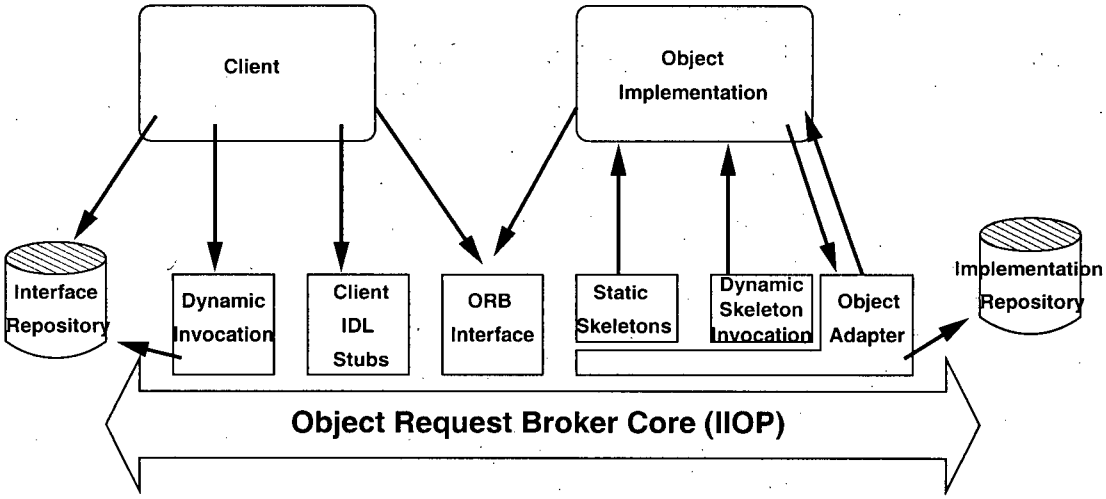

Figure 2.3: ORB structure [23]

2.4.3 CORBA Object Services (CORBAservices)

The term CORBA Object Services refers to fundamental (system-level) object interfaces that extend the functionality of ORB. They are the most commonly used services in building any application. The services provided by the CORBAservices are highly diverse from creating an object, deleting an object from the ORB, and locating an object by its name or properties, to providing operations for monitoring the use of an object. Currently, there are sixteen object services in CORBAservices, as listed below.

The Life Cycle Service: defines the methods for creating, copying, moving, and terminating an object.

The Persistence Service: provides common interfaces to persistently store the state of an object.

The Naming Service: locates an object by object name.

The Event Service: supports the asynchronous communication between objects using events.

The Concurrency Control: provides distributed locks on a given object.

The Transaction Service: provides two-phase commit coordination among objects.

The Relationship Service: provides mechanisms to create, delete, navigate, and manage the dynamic association between objects.

The Externalization Service: provides mechanisms to convert the state of an object into a stream of data and vice versa.

The Query Service: provides query operations for objects.

The Licensing Service: provides operations for monitoring the use of an object.

The Properties Service: provides mechanisms to associate properties to an object.

The Time Service: supports time synchronization in a distributed object environment.

The Security Service: offers a framework for distributed object security.

The Trader Service: matches or locates an object by its properties.

The Collection Service: provides interfaces to create and manage the most common collections.

The Startup Service: starts up an object when an ORB is invoked.

2.4.4 The CORBA Common Facilities (CORBAfacilities)

The CORBA Common Facilities are similar to the CORBA Object Service, but they are at a higher-level and oriented towards end-user applications. They are related to and extended from the existing CORBAservices. The CORBAfacilities is still under construction.

2.5 CORBA-based UPC

In the latter phase of the IWIN project at UBC, CORBA was incoporated into UPC to exploit the advantages of flexibility and versaltality that CORBA's distributed object architecture offered [30, 35]. In CORBA-based UPC, all the entities in the UPC architecture (i.e. UHA, THA, and FA) are presented as distributed objects. These objects move relative to the current location of the user. The Common Object Request Broker Architecture, CORBA, is selected as a middleware layer that provides a homogeneous distributed computing environment independent from the underlying hardware and software technology.

In addition to providing a homogeneous distributed computing environment, CORBA also defines a set of Common Object Services (CORBAservices) and Common Facilities (CORBAfacilities) that provide building blocks for developing any CORBA based application. The CORBAservices support generic and common functionality, such as creating an object, naming an object, and resolving the reference of an object. The CORBAfacilities are the frameworks that provide services directly used by the application objects. Taking advantage of the availability of CORBAservices and CORBAfacilities, the CORBA-based UPC system can utilize some of the core services provided by the CORBAservices, such as the Life Cycle Service, the

Naming Service, the Trader Service, the Security Service, the Persistence Service, and the Relationship Service.

In the CORBA-based UPC approach, terminal mobility is handled by the Mobile-IP. Terminal migration in Mobile-IP is transparent to the TCP and the higher network layers in the OSI model. Since CORBA sits above the TCP/IP layer, terminal migration is also transparent to CORBA. Therefore, the CORBA-based UPC only focuses on the issue of user mobility.

The CORBA-based UPC system consists of the following major objects: the Facility object, the Personal Computing Environment object (PCE), the Terminal Profile object, the User Agent object (UA), the Terminal Agent object (TA), and the Initial Agent object (IA).

2.5.1 Facility object

A facility object is a representation of a computing resource, a network service, or a software application in the system. A facility object is self-descriptive so it can be queried by other objects for supported services and for how to invoke these services. A facility can be either a personal facility, or a shared facility between a group of users. A personal facility is user specific and only available to a specific user at their home network. On the other hand, a shared facility is specialized for a group of users and available on a certain network. To a great extent, a shared facility can be general-purpose and available for everyone, everywhere.

2.5.2 Personal Computing Environment (PCE) object

Each user has a Personal Computing Environment (PCE). A PCE is a set of facility objects that a user wishes to use when they are away from their home network.

A PCE is a persistent object that can be facilitated by the CORBA's Persistence Service. When a user visits a foreign network, their PCE is retrieved and mapped into the facilities that are available at the login terminal (i.e. the Terminal Profile object). The result is a subset of the requested facilities in the user's PCE, referred to as a user's terminal specific PCE.

2.5.3 Terminal Profile object

Each terminal has a terminal profile object kept locally at the terminal that defines the capabilities of the terminal. Both the terminal's physical and software capabilities are specified in the terminal profile object. By Physical capabilities, we refer to the physical properties of a terminal, such as the size of the RAM, the size of the memory in the video card, the monitor mode, and the monitor resolution. Software capabilities refers to the facilities available locally at the terminal.

2.5.4 User Agent (UA) object

Each user has a UA object that acts on behalf of the user in the system. The UA object's main task is to manage a user's PCE object. In addition, the UA object provides interfaces to authenticate a user and to retrieve a user's PCE object once the user is authenticated. The UA object is a persistent object facilitated by CORBA's Persistence Service, and is bound to the user' LUI in CORBA's Naming Services. Thus, an UA object can be located from a user's LUI with the assistance of the CORBA's Naming Service.

2.5.5 Terminal Agent (TA) object:

Each terminal has a TA object that represents the user at the foreign network. The TA object has a reference to the user's terminal specific PCE object (i.e. the subset of a user's PCE that is supported by the current login terminal) which the TA object is responsible to maintain. Moreover, the TA object is also responsible to discover all the facilities that are available at the visiting network, using both CORBA's Name Service and CORBA's Trader Service.

2.5.6 Initial Agent (IA):

An initial agent is an active process that runs at a terminal and provides the start up procedure for users. First the IA authenticates a login user based on CORBA's Security Service. With the assistance of CORBA's Name Service, the IA then finds the user's UA object. After locating the user's UA object, the IA initiates the creation of a TA object and a user's terminal specific PCE object at a terminal.

Figure 2.4, adopted from [30], shows the collaboration of agents in the CORBA-based UPC system.

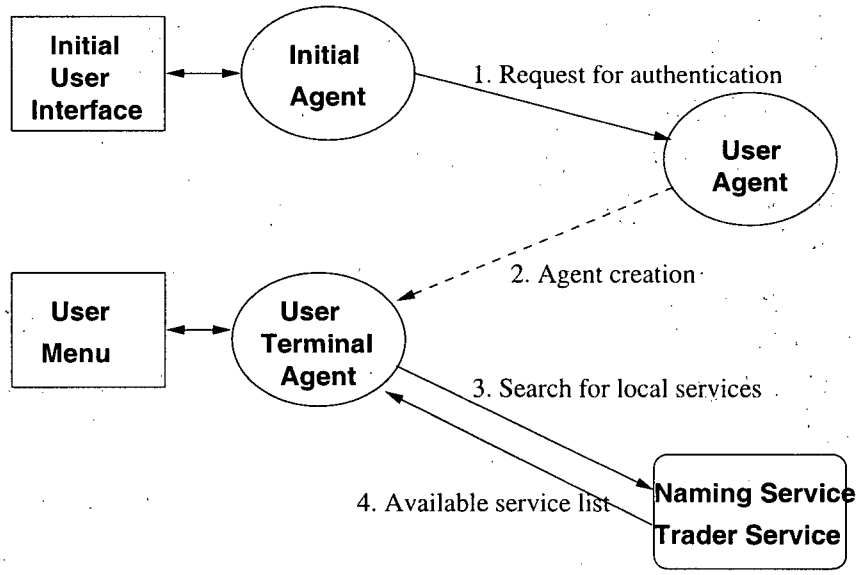

Figure 2.4: Collaboration of agents in the CORBA-based UPC framework [30]

Chapter 3

UPC's Registration and Service Negotiation Protocol (RSNP)

3.1 Overview

In this chapter we present the specification and prototype implementation of UPC's Registration and Service Negotiation Protocol (RSNP). RSNP was specified using the Specification and Description Language (SDL). With the use of SDL, we were able to produce an unambiguous, precise, and concise specification of RSNP. There are several formal specification techniques available to specify and model the behavior of a system, such as Language of Temporal Ordering Specifications (LOTOS) and Estelle Formal Description Technique. However, we chose SDL as the formal specification language for RSNP due to the following reasons:

- SDL is a recognized international standard and accepted by ISO (International Standards Organization). This fact ensures that SDL will be maintained and supported in the future.

- SDL is popular and widely used in the telecommunication industry.
- The graphical SDL is intuitive and easy to work with. It clearly displays the relationships between blocks, processes and how they interact.
- The nature of SDL is suitable for specification of a communication protocol since the SDL process is described as Extended FSM. The FSM is an excellent model for any communication protocol design.

In regards to the SDL tool, we used the tool developed at the KFKI Research Institute for Measurement and Computing Techniques in Hungary. This SDL tool is a part of a tool set called PROCONSUL (i.e. PROtocol CONSULtant) and consists of a SDL simulator and language sensitive editor that supports the grammar of the SDL 1992 version. The editor has a syntax checker, which helps to create syntactically correct SDL specification. The SDL simulator represents a SDL system as a block diagram, and displays the output of the simulation in the form of a time diagram or a message sequence chart (MSC). We can step through the simulation execution and simulate different scenarios by changing a variable value, changing the delay time for a channel, and even adding to or deleting signals from the signal queue. For example, we can simulate a loss of message by removing it from the signal queue, or we can simulate the protocol behaviour during a network congestion period by simply increasing the delay time in a channel. With the use of the SDL simulator we checked for RSNP specification's completeness and correctness. In addition to this, we were able to simulate alternative solutions for any problems that come up.

3.2 UPC Protocol Architecture Modification

RSNP is specified with several modifications to the architecture as well as additions of new entities, as compared to the UPC protocol architecture mentioned in section 2.3. The modifications and their justification are as follows:

- Terminal related information such as the terminal identity and the terminal profile are kept locally at the terminal. The terminal identity is the permanent IP address of a terminal. The terminal profile is basically the terminal configuration information, such as resident operating system, file format, and monitor display mode, and so on. This information is currently available at any terminal so there is no need to replicate the information at the THA. This alleviates the need for database management in the THA.

- We extended the terminal profile to store information about software applications locally available at this particular terminal, such as the name of the application, the type of the application, the path of the executable file of the application, and the mapping information for a mobile user's preference and configuration files. The mapping information maps a mobile user's preference and configuration file for an application on the home network, to the preference and configuration file for an application available on the foreign terminal.

- Since the terminal profile is kept locally at the terminal, there is no need for a Terminal Home Agent.

- We introduce a new agent, called Mobile User Agent. This agent runs locally at a terminal and provides a login interface to users. The Mobile User Agent is responsible for registering, re-registering when a connection lifetime expires,

or re-registering when the user and terminal move to a different network. It is also responsible for producing a Session Service Profile that restrains the user from unregistered services.

- Each foreign network has a Service Profile that specifies the computing resources and network services that are available at this particular network, and the personal applications that are allowed to execute while the mobile user is visiting the network. The UFA provides access to this Service Profile during the service negotiation phase.

- We introduce a temporary service profile located at a terminal, called User-Terminal Service Profile. This profile has the same format as any other service profile and is created after a successful registration. The User-Terminal Service Profile is the intersection between the User Service Profile and the Terminal Service Profile, and is created once for each login session. This service profile exists because the binding between a mobile user and mobile terminal during a login session is fixed. As a result, the User Service Profile does not have to be downloaded every time the mobile user and their mobile terminal experiences a hand-off with an ongoing session.

- We introduce a temporary service profile located at a terminal, called Session Service Profile. This profile is specific to a particular valid login session within a particular visiting network. It is the intersection between the User-Terminal Service Profile and the UFA Service Profile. The login user's access is limited to the services that only exist in the Session Service Profile. This profile is recreated every time a mobile user and their mobile terminal move to a new foreign network.

3.3 SDL Specification of RSNP

At the top level, we can describe the RSNP in UPC as a system that accepts the login and logout request from a user in the environment and then outputs the status message to the user. This system consists of one block called MobileIP, one block called Mobile User (MU), a set of two similar blocks of type User Foreign Agent (UFA), and another set of two similar blocks of type User Home Agent (UHA). Figure 3.1 presents the System Diagram of the UPC's Registration and Service Negotiation Protocol. (Please note this diagram is for illustration purpose only, it is not the system diagram that is directly printed out from the SDL's graphical representation.)

- The MU block is connected to the environment through a channel called User, to the Mobile-IP block through a channel called MIP, and to the UFA block through a channel called MU_UFA.
- The UFA block is connected to the UHA block via a channel called UFA_UHA, and to the environment via a channel called UFA_Debug.
- The UHA block is connected to the environment through a channel called UHA_Debug.

On each channel in figure 3.1, there is a list of SDL signals that are carried by the channel in the specified direction.

The User channel: this channel carries the SDL's signals between the environment (i.e. a user) and a MU block. The SDL signals carried in this channel are as follows:

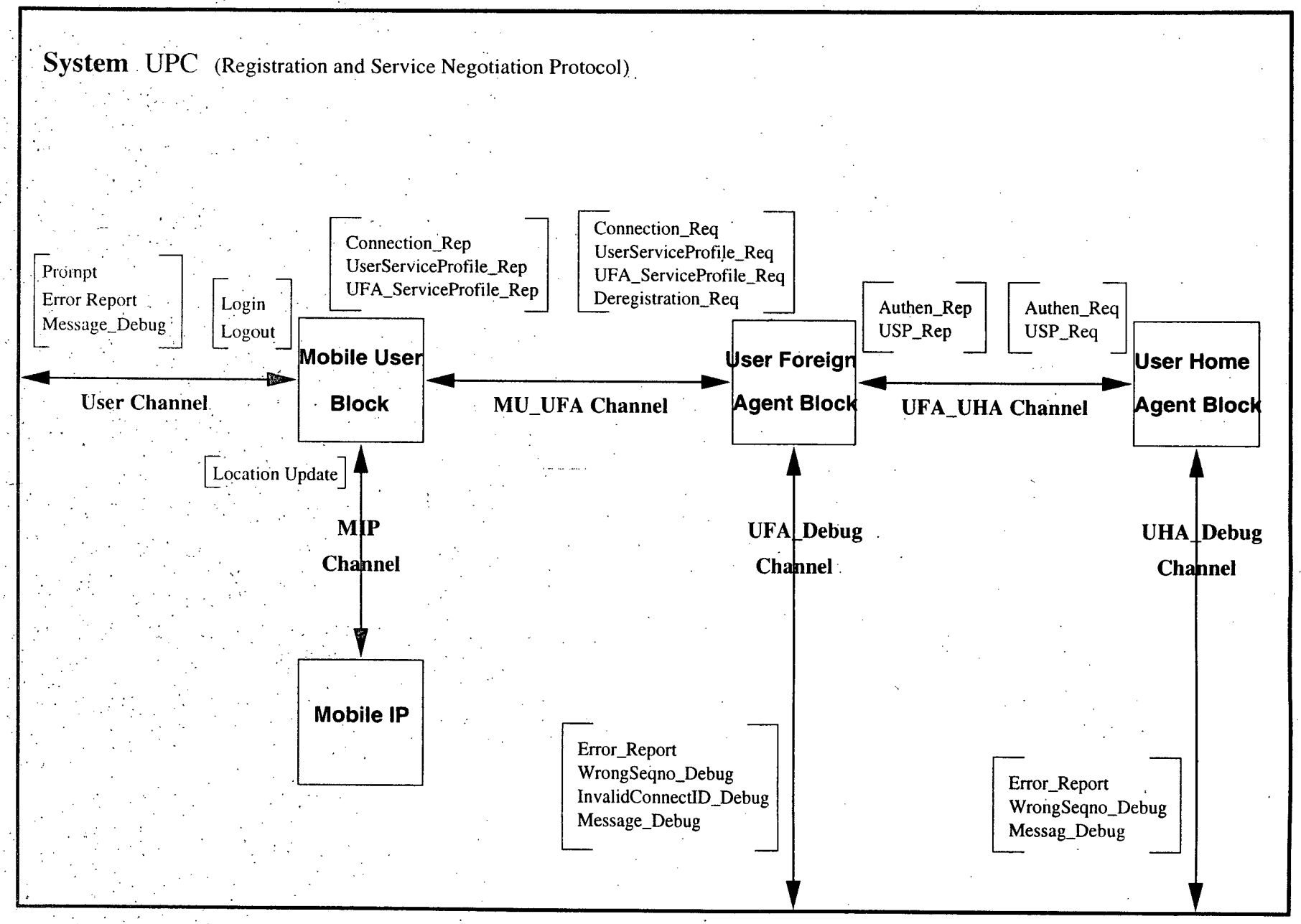

Figure 3.1: System Diagram of UPC's Registration and Service Negotiation Protocol

Login signal: this signal describes a login request from a user with the input parameters, such as the user's LUI, the user's password, and the IP address of the user's UHA.

Logout signal: this signal describes a logout request from a user who has a valid login session.

Prompt signal: this signal describes a prompt for user inputs.

Error Report signal: this signal describes a system response when an error occurs.

The MIP channel: this channel carries the SDL signals between a MIP block and a MU block.

Location Update signal: this signal describes the location update message that Mobile-IP sends to the MU, informing it that the terminal is moving into a new foreign network. It has the new UFA IP address as the input parameter.

The MU_UFA channel: this channel carries the SDL signals between a MU block and an UFA block. The SDL signals carried in this channel are as follows:

Connection Request signal: this signal describes a request from a MU to an UFA for a connection. It has the user's LUI, the user's password, the user's UHA IP address, a request connection lifetime, and a sequence number as the input parameters.

Connection Reply signal: this signal describes a reply from an UFA to a MU for a connection request. It has a returned code, a string message, a granted connection lifetime, a connection ID, and a sequence number as the output parameters.

User Service Profile Request signal: this signal describes a request from a MU to an UFA for a user's service profile. It has the user's LUI, a connection ID, and a sequence number as the input parameters.

User Service Profile Reply signal: this signal describes a reply from an UFA to a MU for a user service profile request. It has a returned code, a string message, a service profile, and a sequence number as the output parameters.

UFA Service Profile Request signal: this signal describes a request from a MU to an UFA for the UFA's service profile. It has the user's LUI, a connection ID, and a sequence number as the input parameters.

UFA Service Profile Reply signal: this signal describes a reply from an UFA to a MU for an UFA's service profile request. It has a returned code, a string message, a service profile, and a sequence number as the output parameters.

De-registration Request signal: this signal describes a request for de- registration of a valid established connection. It has a connection ID as the input parameter.

The UFA_UHA channel: this channel carries the SDL signals between an UFA block and an UHA block. The SDL signals carried in this channel are as follows:

Authentication Request signal: this signal describes a request from an UFA to an UHA for authenticating a user. It has the user's LUI, the user's password, and a sequence number as the input parameters.

Authentication Reply signal: this signal describes a reply from an UHA to UFA for an authentication request. It has a returned code, a string message, and a sequence number as the output parameters.

User Service Profile Request signal: this signal describes a request from an UFA to an UHA for a user's service profile. It has the user's LUI and a sequence number as the input parameters.

User Service Profile Reply signal: this signal describes a reply from an UHA to an UFA for a user's service profile request. It has a returned code, a string message, a service profile, and a sequence number as the output parameters.

The UFA_Debug and the UHA_Debug channel: these two channels carry SDL signals between the UFA block, UHA block and the environment respectively. These signals are mainly used to display the debug and status information at both the UFA and UHA.

The MIP channel: this channel carries the SDL signals between a MU block and a Mobile-IP block. There is only one signal, called Location Update, on this channel. This signal has the new UFA's IP address as the input parameter.

3.3.1 Pseudo Mobile-IP Block

The Mobile-IP block describes a pseudo mobile-IP. Figure 3.2 is the block diagram of the Mobile-IP block. The Mobile-IP block contains one single process called MIP_Pseudo that simulates the movement of a mobile terminal. After detecting the terminal has moved to a foreign network, it informs the MU block about the change using a location update signal.

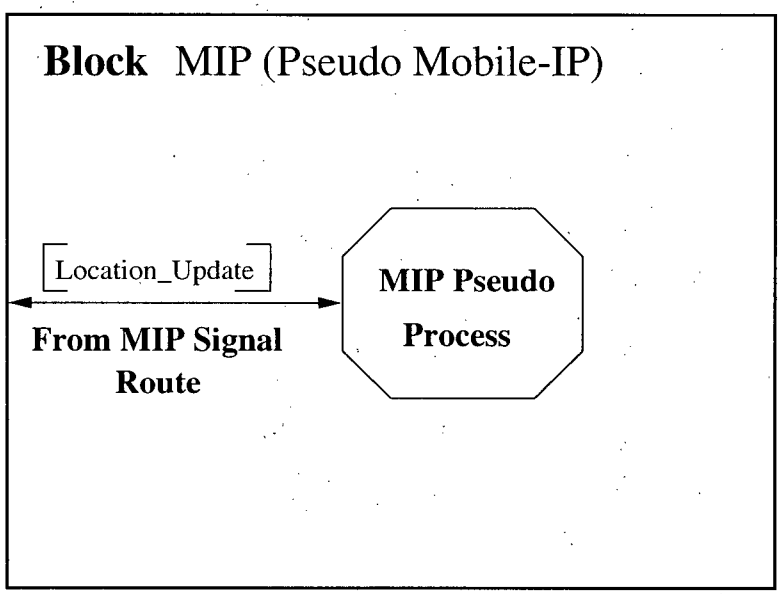

Figure 3.2: Block Diagram of a Pseudo Mobile-IP

3.3.2 Mobile User Block

The MU block describes a Mobile User Agent as mentioned in section 3.3. This block provides the interfaces for user inputs, such as login and logout requests. Figure 3.3 illustrates the block diagram of the MU block.

This block contains one process called MU_Process whose behaviour is described in the attached extended FSM, in figure 3.4. The MU_Process consists of four states: the Idle state, the Registering state, the Registered state, and the Negotiation state.

Idle state: in this state, the MU_Process is ready to accept a login request from a user.

Registering state: the MU_Process enters this state after receiving a login request from a user or a location update message from the Mobile-IP. During this state, the MU_Process contacts the UFA to request a connection and waits for

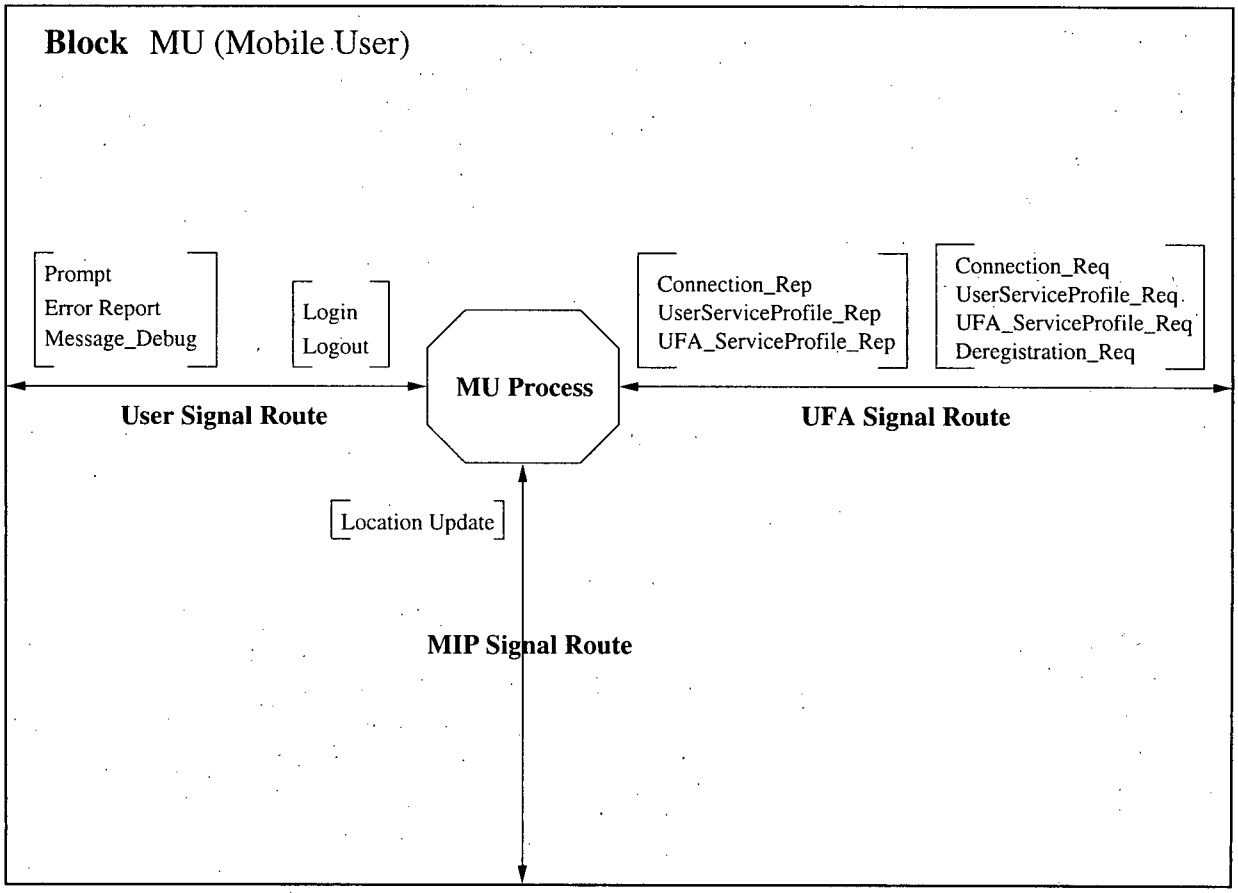

Figure 3.3: Block Diagram of a Mobile User

a reply.

Registered state: the MU_Process enters this state after receiving a connection acceptance from the UFA. It waits for user requests that include execute an application or perform network services.

Negotiation state: the MU_Process enters this state after sending requests for the user's service profile and the UFA's service profile and awaits the replies.

3.3.3 User Foreign Agent Block

The UFA block describes a UFA. Figure 3.5 illustrates the block diagram of the UFA block.

There is one process in the UFA block called UFA_Process, whose behaviour is described in the attached extended FSM, in figure 3.6. The UFA_Process consists of four states: the Idle state, the Process Connection Request state, the Session Established state, and the User Service Profile Request Processing state.

Idle state: in this state, the UFA_Process is ready to accept a connection request from the MU.

Process Connection Request state: in this state, the UFA_Process processes the connection request from the MU. It contacts the specified UHA to authenticate the login user and awaits an authentication reply.

Session Established state: the UFA_Process enters this state after it grants a connection to the MU. In this state, it waits for further requests from the MU, such as a request for using network services.

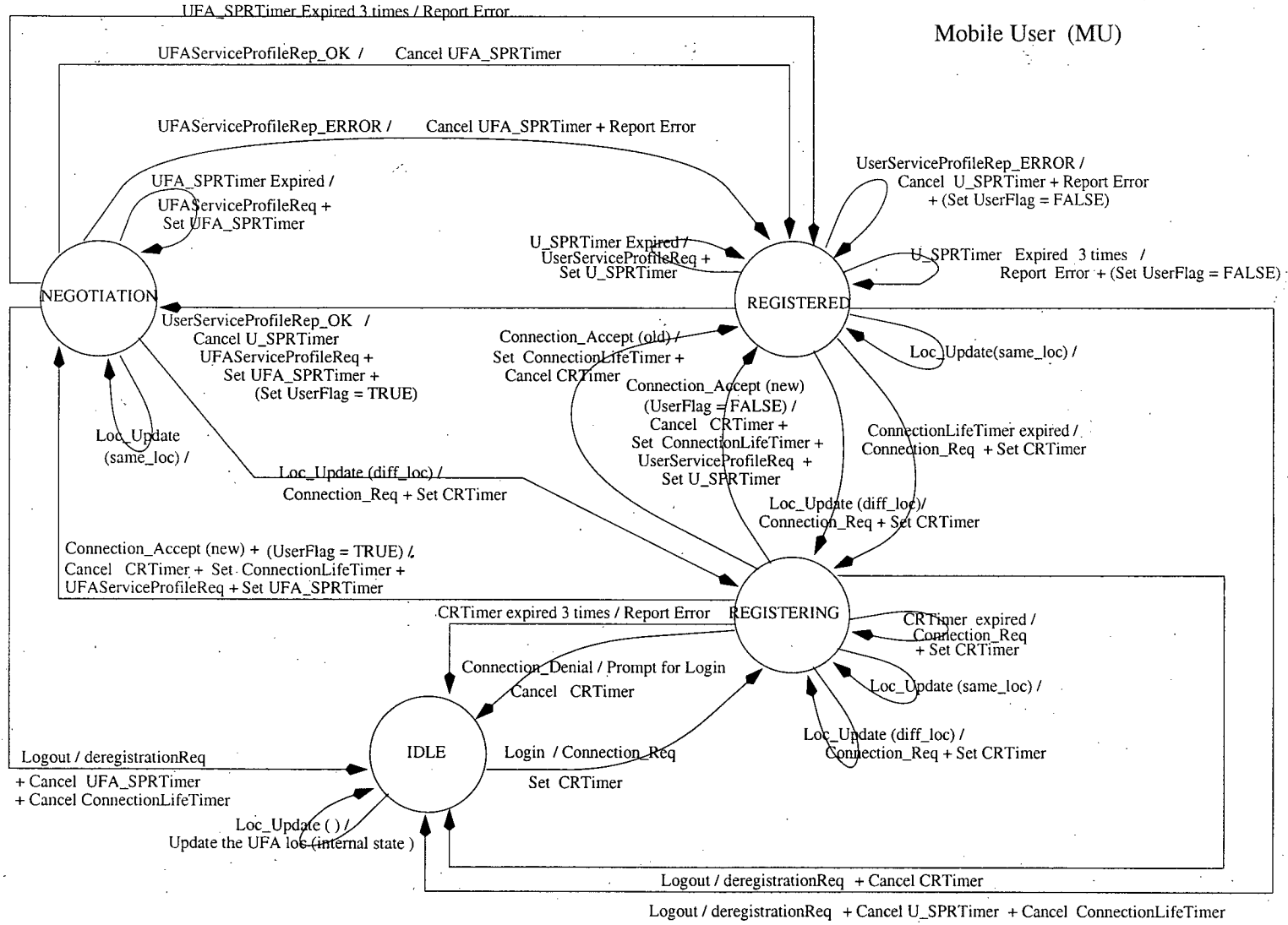

Figure 3.4: Mobile User Agent's Extended Finite State Machine

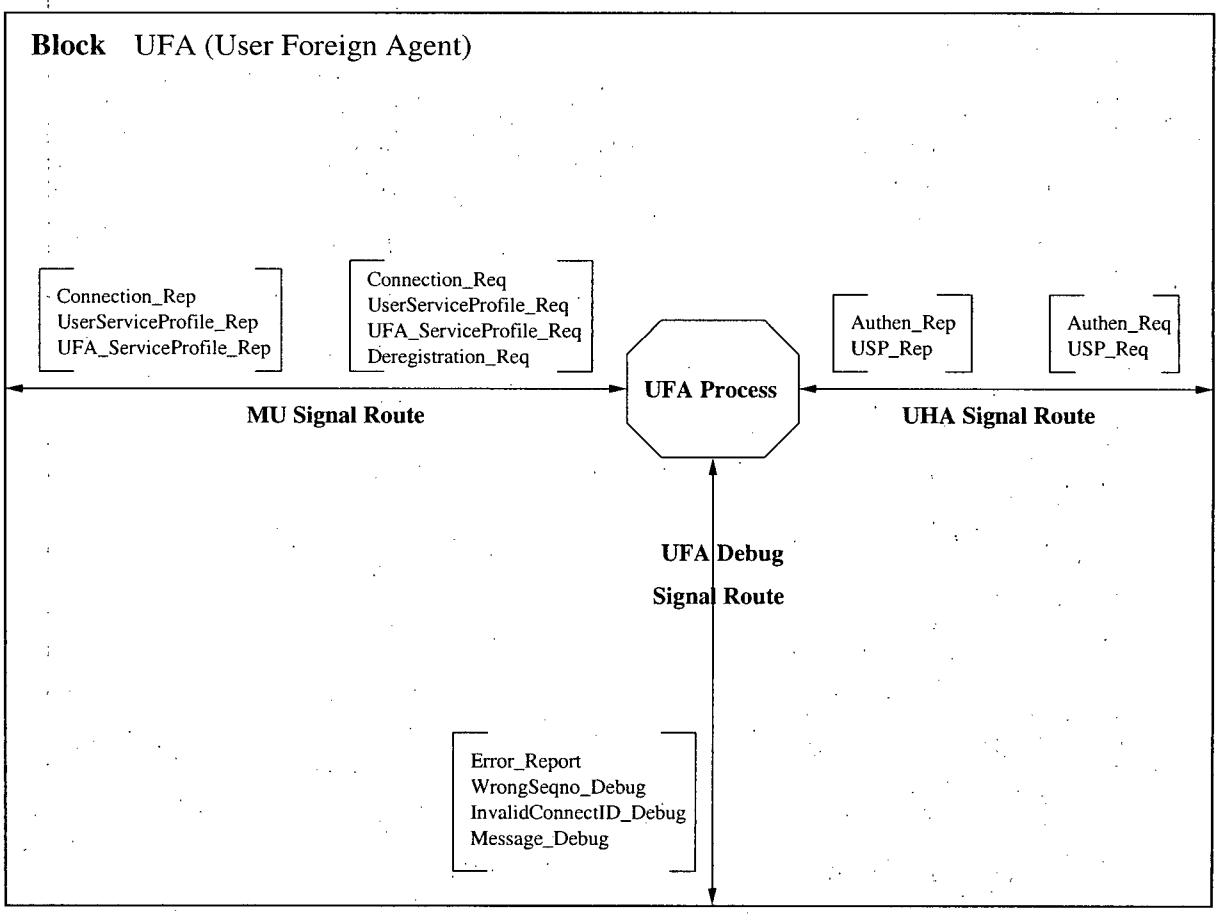

Figure 3.5: Block Diagram of a User Foreign Agent

User Service Profile Request Processing state: in this state, the UFA_Process processes the request for a user service profile from the MU. It contacts the user's UHA to request the user's service profile and awaits the reply.

3.3.4 User Home Agent Block

The UHA block describes a UHA. Figure 3.7 diagrams the UHA block.

There is one process in the UHA block called UHA_Process, whose behaviour is described in the attached extended FSM, in figure 3.8. The UHA_Process consists of three states: the Idle state, the Checking ID and Password state, and the Service Profile Request Processing state.

Idle state: in this state, the UHA_Process is ready to accept any requests from the UFA.

Checking ID and Password state: the UHA_Process enters this state after receiving a request for user authentication.

Service Profile Request Processing state: the UHA_Process enters this state after receiving a request for a user's service profile.

3.4 Prototype Implementation of UPC

After having a concise specification, we implemented a prototype of RSNP on Sun Microsystem Solaris 2.x using C as the programming language. The user graphical interface was written in Tcl/Tk. The implementation code space is roughly about 8000 lines. The prototype of RSNP is subjected to the following assumptions and requirements:

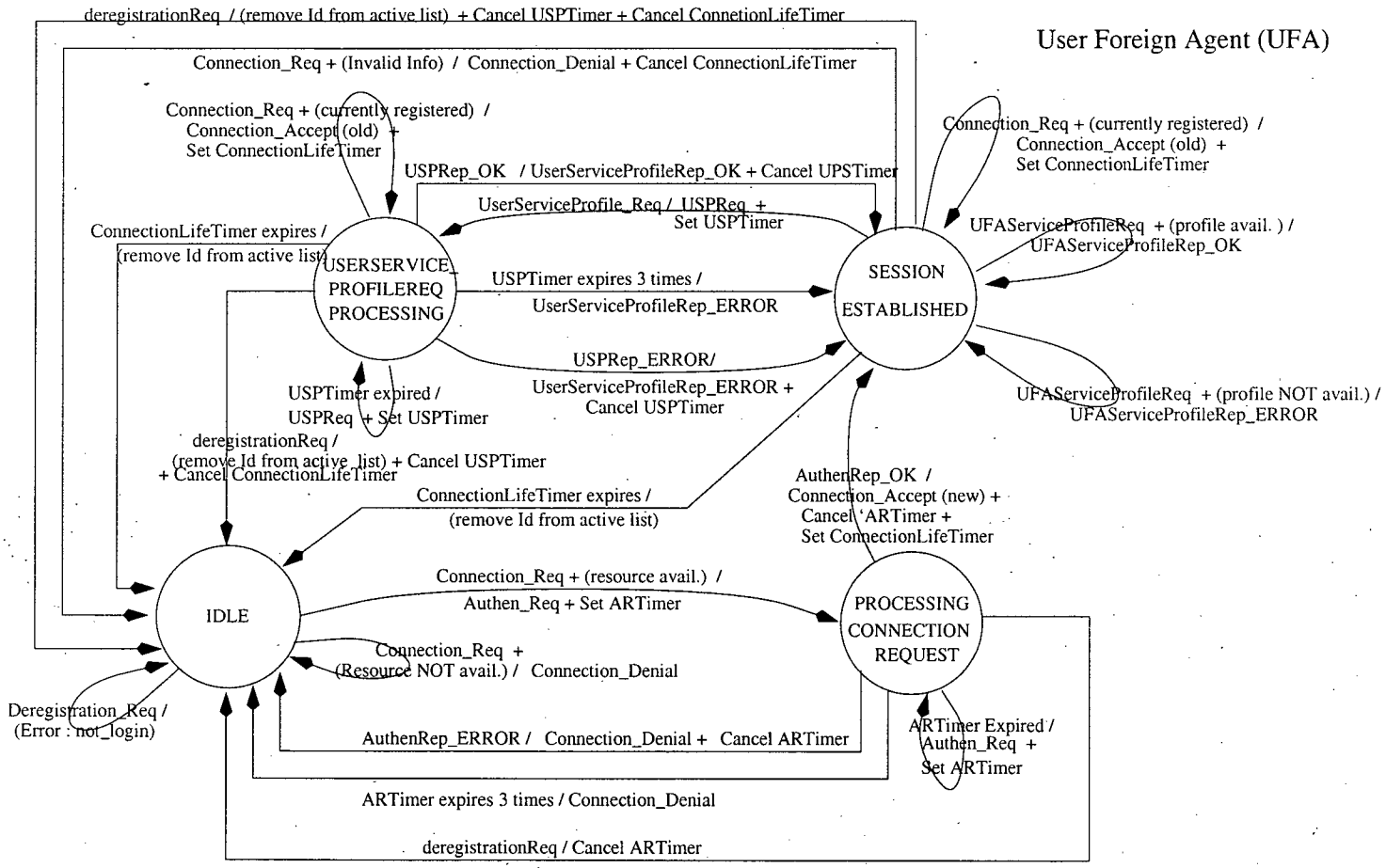

Figure 3.6: User Foreign Agent's Extended Finite State Machine

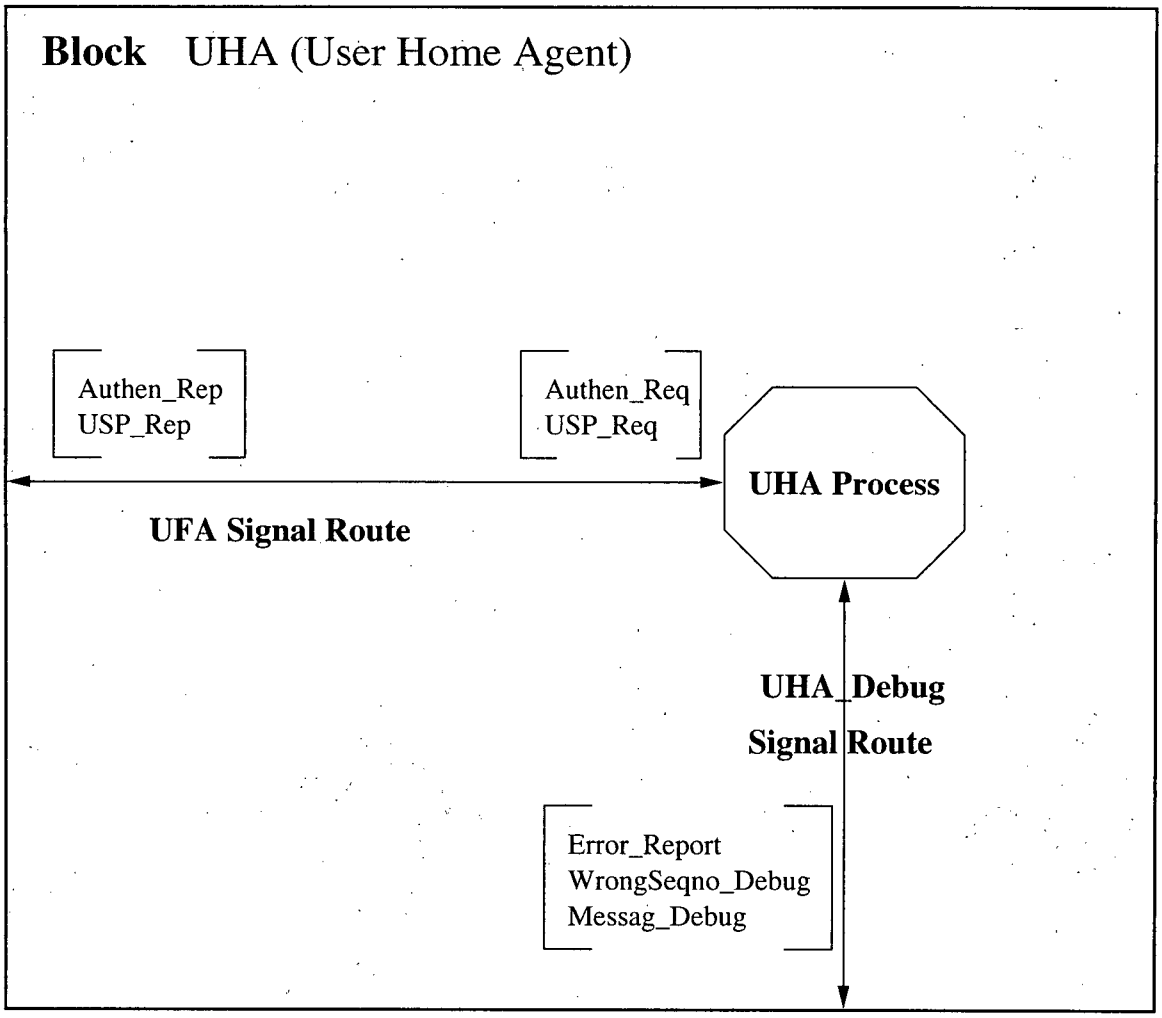

Figure 3.7: Block Diagram of a User Home Agent

Figure 3.8: User Home Agent's Extended Finite State Machine

3.4.1 Assumptions and Requirements:

- The applications supported here are limited to internet applications such as web browsers (e.g. Netscape, Mosaic), news clients , mail clients, and so on.
- This prototype implementation does not support multi-platforms. The supported platform considered here is UNIX.
- A local user whose login mobile terminal comes from a foreign network is considered a visiting user since the mobile terminal may not support all the services that a fixed terminal in this network supports.
- The UFA and the Mobile-IP Foreign Agent (FA) are required to operate on the same machine, or in other words, they should have the same IP address. This requirement makes the agent (UFA) discovery procedure in the UPC efficient and simple. As a mobile terminal (MT) and a mobile user enter a foreign network, the Mobile IP detects the move and informs the MU of the current UFA's IP address (this should be the same as the FA IP address). The Mobile-IP need to be modified to provide this location update service.
- A mobile user is required to know the UHA's location, whether it is a UHA's IP address or the name of the host that UHA is currently running on. The UHA location can be extracted from the LUI, but this requires that the UHA resides on the same host as the internet mail server. It is unsafe to run any user program on the mail server's host, and to the extent that this is a prototype implementation, this requirement is reasonable. If the user is local and the terminal is also local, then this requirement is waived.

3.4.2 Registration and Service Negotiation Procedure

The behaviour of the RSNP depends on the type of terminal, fixed or mobile. There is no hand-off and UFA location update procedures for a fixed terminal. For a mobile terminal, by default, the Mobile User (MU) is configured to know the location of the UFA in its home network.

The following is the sequence of events during the Registration and Service Negotiation:

- After receiving the user's information, such as the LUI, the password, and the UHA's IP address, the MU at the login terminal verifies whether the user is local or foreign by comparing the domain portion of the LUI and the current UFA's IP address. The terminal's type, fixed or mobile, is identified from the terminal configuration. If the user is local in this administrative domain and the terminal is fixed, the MU allows this user to login locally. The user has full access to any services and applications available. If the user is from a foreign administrative domain, or the user is local in this administrative domain but the terminal is a foreign terminal, the MU contacts UFA for a connection request.

- After receiving a connection request from a MU, the UFA checks its current status, i.e. number of connections available. If the UFA is able to handle another connection, it contacts the user's UHA for user authentication.

- The user's UHA validates the requested LUI and its corresponding password. If both the LUI and password are valid, the UHA returns a valid authentication reply to UFA.

- As the UFA receives a valid authentication reply from the user's UHA, it generates a connection ID and sends a connection acceptance message along with this connection ID to the MU.

- After the registration succeeds, the MU initiates the service negotiation procedure by sending a request for the User Service Profile to the UFA. The UFA forwards this request to the user's UHA. The reply comes to the UFA and is then forwarded to the MU.

- If the MU receives a valid reply, it creates a User-Terminal Service Profile by intersecting the User Service Profile with the Terminal Service Profile.

- If the User-Terminal Service Profile is not empty, the MU contacts the UFA to retrieve the UFA's Service Profile and then produces a Session Service Profile by intersecting the UFA's Service Profile with the User-Terminal Service Profile.

3.4.3 Re-registration after Expiration of Connection Lifetime

Each connection granted from the UFA has a unique ID and a limited lifetime. A connection becomes invalid if the connection lifetime at the UFA expires. To maintain a connection, the MU has to send a re-registration request whenever its connection lifetime expires. The re-registration procedure after expiration of a connection lifetime is very simple compared to the re-registration procedure after a hand-off. When a connection lifetime expires, the MU sends a re-registration request with the associated connection ID and requested desired lifetime. If the connection ID is valid, the UFA updates the connection lifetime for this connection ID and then sends a reply back to MU.

3.4.4 Re-registration after Hand-off

Hand-off only occurs with a mobile terminal. The hand-off in the mobile terminal is handled by the Mobile IP. The MU detects a move into a foreign network when it receives a location update message from the Mobile IP and the UFA's IP address in the location update message is different from its current connected UFA's IP address. If the MU is serving an ongoing session, it performs the re-registration procedure. The MU contacts the new UFA and provides it with all the user's information such as the LUI, the password, and the UHA's IP address. To authenticate the user, similar steps as used in the registration procedure are carried out. The ongoing session is terminated if either the new UFA is not available for any services or authentication of the user failed.

After successful registration with the new UFA, the MU contacts the new UFA to retrieve the new UFA's Service Profile. A new Session Service Profile is created from intersecting the User- Terminal Service Profile and UFA's Service Profile. Current running services or applications that do not exist in the Session Service Profile are considered illegal and are immediately terminated by the MU.

3.4.5 Datagram Format used in Registration and Service Negotiation Protocol

Communication between MIP and MU

This is the datagram format for the location update packet from Mobile IP.

- UDP fields
  - Source Port: variable
  - Destination Port: 9999

- The UDP Header
  - Type: Location Update
  - Location: IP address of UFA

Communication between MU/UFA or UFA/UHA

There is only one header type used for all the request and reply packets between MU and UFA, or UFA and UHA. User ID, password, and service profile are inserted into the data portion after the header.

- UDP fields
  - Source Port: variable
  - Destination Port: copied from the source port of the corresponding Request or
    - from MU to UFA use 8888
    - from UFA to UHA use 6666
    - from UHA to UFA use 7777
- The UDP Header
  - Type:
    - Connection Request
    - Connection Reply
    - Authentication Request
    - Authentication Reply
    - User Service Profile Request

    * User Service Profile Reply
    * UFA Service Profile Request
    * UFA Service Profile Reply

- Code: A value indicating the result of the request. It is set to 0 in the request packet. Possible errors are:
    * For Connection Request
        - Registration Accepted
        - Registration Denied
        - Unspecified Reason
        - Insufficient Resources
        - Authentication Failed
        - Requested Lifetime Too Long
        - Poorly Formed Request
        - Unknown UHA Address
        - UHA host is Unreachable
        - UHA port is Unreachable
    * For Authentication Request
        - Authentication OK
        - Authentication ERROR:
        - Unspecified Reason
        - Identification Mismatch
        - Poorly Formed Request
        - Un-Registered User

* For User Service Profile Request
    - OK
    - ERROR, Unspecified
    - UHA Unreachable
* For UFA Service Profile Request
    - OK
    - ERROR, Unspecified
    - UFA Unreachable

- Lifetime: The number of seconds remaining before the registration is considered expired. A value of 0xffff indicates infinity. It is set to 0 for the reply packet with an error code.
- UHA: IP address of the User Home Agent
- Seqno: the sequence number of this packet.
- ID: the connection ID of this connection. It is set to 0 in packet type Connection Request, except in the case of re-registration after a connection lifetime expires. In this case it is set to the actual connection ID.
- Data length: the length of the data following the header.

Chapter 4

CORBA-based UPC Prototype Implementation

4.1 CORBA-based UPC Prototype Implementation

In this chapter, we present the implementation of the CORBA-based UPC prototype. As mention in section 2.5, CORBA-based UPC architecture is an alternative approach in modeling the UPC concept. In this architecture, the UPC components are modeled as distributed objects with CORBA as the object bus that facilitates the interaction, integration, and distribution of these objects. We implemented the prototype using Java as the programming language, Visigenic's VisiBroker for Java as the CORBA framework, and Sun Microsystem's Solaris 2.x as the computing platform. The prototype code space is approximately 5000 lines.

Because the implementation is a prototype and there are insufficient computing resources, the prototype is implemented based on the following simplifications and requirements:

- The UPC architecture utilizes a wide range of services defined by the CORBA Object Services, such as the Naming Service, the Trader Services, the Life Cycle Services, and the Security Services. However, some implementations of these services (i.e. the Trader Services) were not available at the time we implemented the UPC prototype. As a result, the lack of CORBA services has influenced our implementation of the prototype to a certain extent. The service discovery and service negotiation procedure is simplified to a direct mapping in the application name and configuration name. Thus, the terminal profile is simpler than the terminal profile proposed in Zhu, Toeroe, Leung, and Vuong[35]. On our implementation the terminal profile defines only the software capability of the terminal, with no information regarding the terminal's physical capability.

- The terminal must support Java and CORBA's ORB. In addition, there must be enough memory in the terminal to download the application client class file and the application configuration file.

As the first step in the implementation, we defined all the UPC objects' interfaces using the Interface Definition Language (IDL). An overview of IDL was presented in section 2.5. Then, we ran the IDL files through CORBA compliant pre-compiler to generate the stubs and skeletons, which were used as a frame in implementing an object. The following sections present the IDL interface definition for all the major objects as well as their functionalities in the CORBA-based UPC architecture, such as Personal Computing Environment (PCE), User Home Agent (UHA), Terminal Profile, Terminal Agent (TA), and Initial Agent.

4.1.1 Personal Computing Environment (PCE)

Each user has a PCE which is kept and maintained at the User Home Agent. The PCE is designed to specify enough information so that the Terminal Agent can untilize the CORBA Trader Services to discover and negotiate the services available for a user while they visit a foreign network. The IDL code of the PCE is attached in Appendix A.

The PCE is a list of service structures, each of which is composed of:

- Service Name: the name of a service object, defined as a string.
- Service Type: the type of service object, i.e. an application client object, application configuration, or network resources.
- Service Object: the reference to a service object. This object can be an application client object, an object that allows access to an application configuration, or an object that allows access to network resources.
- Trade Information: a structure that stores all the necessary information needed for the CORBA Trader Services to discover or locate a service, i.e. the service type, the service properties, the constraint of the search and the search policies.
- Hardware capability: a list of the requirements about physical hardware support for this service. For example, to play a midi file, a terminal is at least required to be equiped with a sound card and speakers.

4.1.2 User Home Agent (UHA)

For each administrative domain, there is a UHA. The UHA manages the list of users registered in its administrative domain, and the users' corresponding PCEs. The

IDL code of the UHA is attached in Appendix A.

The UHA exports the following services:

- boolean tokenauthent(in string LUI, in string password, out Token token):
  This method authenticates a user identified by the LUI and the password. Upon successful authentication, a token (or a validation ID) is returned to the caller.

- inform_location(in string LUI, in string address):
  This method informs the UHA of the current location of the user (i.e. IP address of the terminal).

- PCE::Profile getprofile(in string username):
  This method retrieves a user's PCE.

- string download(in Token token, in string filename):
  This method downloads a file from the user's home network.

- boolean upload(in Token token, in string filename, in string data):
  This method uploads a file from the TA to the user's home network.

- boolean updateProfile(in string username, in PCE::Profile p):
  This method allows a user to modify their PCE.

- void logout(in Token token):
  This method informs the UHA that a user is logging out.

4.1.3 Terminal Profile

Each terminal has a terminal profile which is stored in a file at the terminal. In our prototype, the terminal profile is implemented as a JAVA class that maintains a list

of the software application names that are available at the terminal (e.g. Netscape, Mosaic, and Internet Mail). These software application names determine which software applications are available at a terminal to a user when they login to this terminal.

In addition to the application name, for each software application, the following corresponding information is kept in the terminal profile:

- The full path executable file name of the software application (e.g. for Netscape, the file name is /usr/application/netscape.exe). This full path executable file name specifies where to fire up the software application.

- A list of configuration names and preference names (e.g. bookmarks) and the corresponding full path configuration file names and preference file names (e.g. /usr/application/netscape/bookmarks.html). The list of configuration names specifies and limits the ability of a visiting user to replace these configurations with their own configuration from their home network. When a configuration file is downloaded from a user's home network, the configuration name and configuration file name are used to map the downloading configuration file to the local configuration file at the terminal.

In order to aid in the generating and updating of a service profile, we implemented a utility application with GUI, called Service Manager. The Service Manager utility is available to the system administrator or the owner of the portable terminal. The Service Manager utility supports the following functions:

- Add a new application name and its executable file name into the terminal profile.

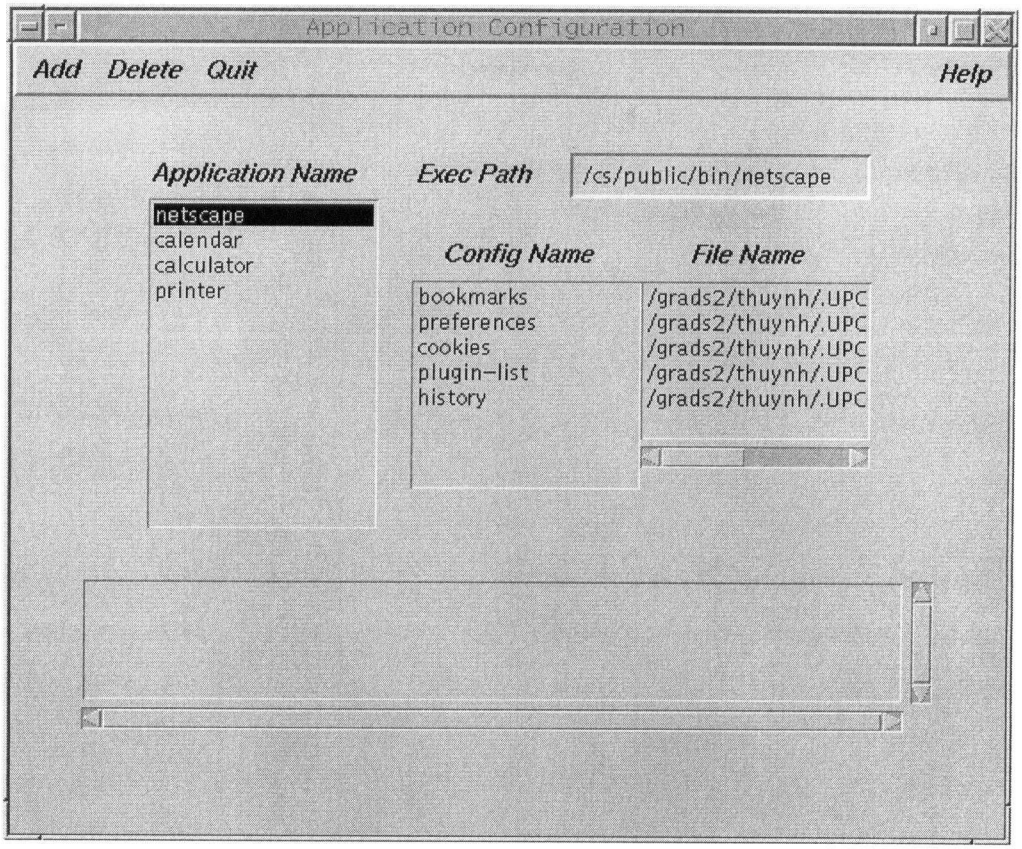

Figure 4.1: Main screen of the Service Manager utility

- Add a new configuration name and its configuration file name to an existing application in the terminal profile.
- Delete an existing configuration name and its configuration file name from an existing application in the terminal profile.
- Delete an existing application name and its corresponding executable file name, the configuration name list and configuration file name list from the terminal profile.

A screen shot of the Service Manager is illustrated in Figure 4.1.

4.1.4 Terminal Agent (TA)

The TA is implemented as an application server for the Initial Agent. Residing locally at a terminal, the TA communicates with the login user's UHA to perform the following tasks:

- Process a login request for a user.
- Retrieve the user' PCE from their home network.
- Perform service discovery and negotiation. This process produces a set of network services and applications requested by the user and which are also available at the login terminal.
- Download the configuration or preference files that a user specifies for a particular application when this application is locally executed.
- Download the class files of the client of the application that is remotely executed.
- At shutdown, synchronize the downloaded configuration or preference files with copies at the home network.

The following describes the interfaces supported by TA. These interfaces are defined using OMG's IDL. Refer to Appendix A for the entire IDL code.

- boolean login(in string LUI, in string password) raises (Fail):
  This method uses the domain name portion given in the LUI and the CORBA Name Services to resolve the object reference of the login user's UHA. After obtaining the UHA object reference, this method then invokes the `authenticate()`

method exported by UHA to authenticate the user. If the authenticate operation is successful, this method invokes the `getprofile()` method from the UHA to get a reference to the user's PCE. This method compares the user's PCE with the Terminal Profile to produce a list of network services and applications specified by the user in the user's PCE which are also available at the login terminal. This method also extracts from the PCE all the network services and applications that are remotely available. "Remotely available" network services or applications are network services that can be executed at the home network or applications that are implemented using the CORBA Client/Server framework.

- void logout() raise (Fail):

  This method first closes all the running network's services and applications. Secondly, for local applications, if any of the configuration files downloaded from the user's UHA had been modified, these files are synchronized with copies at the user's UHA by invoking the `upload()` method exported by the UHA. And finally, this method deletes both the configuration and class files downloaded during this login session.

- string runLocalApp(in string appName) raise (Fail):

  This method executes a local application. If there is a configuration file specified with this application in the user's PCE, this method invokes the `download()` method exported by the UHA to download this configuration file.

- string runRemoteApp(in string appName) raises (Fail):

  This method executes a remote application. It first downloads the "client" of the application from the UHA, as specified in the user's PCE, then instantiates

the object and runs it.

- string getLocalServiceList() raises (Fail):

  This method returns a list of network services and applications specified by the user in the PCE which are also available at the foreign terminal.

- string getRemoteServiceList() raises (Fail):

  This method returns a list of network services that can be executed at the home network and applications that are implemented using the CORBA Client/Server framework.

4.1.5 Initial Agent

The Initial Agent is an active process executed at a terminal. It is responsible to accept inputs from users and output the responses from the UPC system. The IA is implemented as a TA's CORBA client application. The IA invokes the service provided by the TA corresponding to the user input.

The IA is implemented as a Java class that consists of two inner classes called Login Window class and Application Manager class.

- The Login Window class provides a screen where a user can make a login request by entering their LUI and password. The screen shot of the Login screen is shown in the Figure 4.2.

- The Application Manager class provides a window where a user can select to execute a software application or request a network service. There are two selectable lists of software applications and network services that a user is granted to access. The first list displays all the network services that are available at the visiting network and software applications that are locally

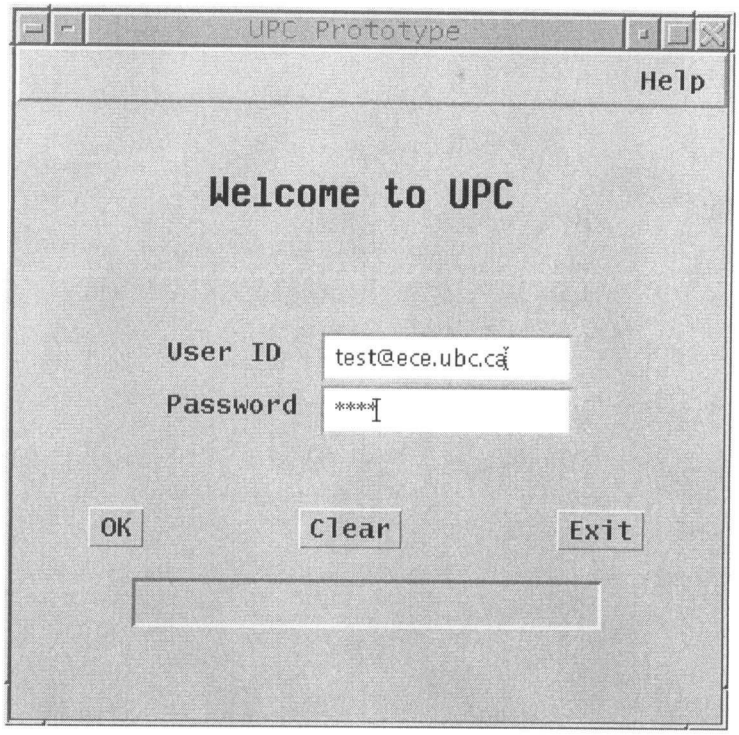

Figure 4.2: Login screen of the UPC

available at the login terminal, and the second list displays all the network services and software applications that are available at the home network which can be remotely executed. The screen shot of the Application Manager screen is shown in Figure 4.3.

After a user enters their LUI and password, the IA instantiates a TA object and invokes the `login()` method exported by the TA object. If the login request is successful, the IA closes the login window and displays the application manager window. The IA obtains the list of locally available network services and applications and the list of remotely available network services and applications by invoking the `getLocalServiceList()` and `getRemoteServiceList()` method respectively from the TA. These two lists are displayed in the window manager so that a user can easily select to execute the desired network service or application. Depending on

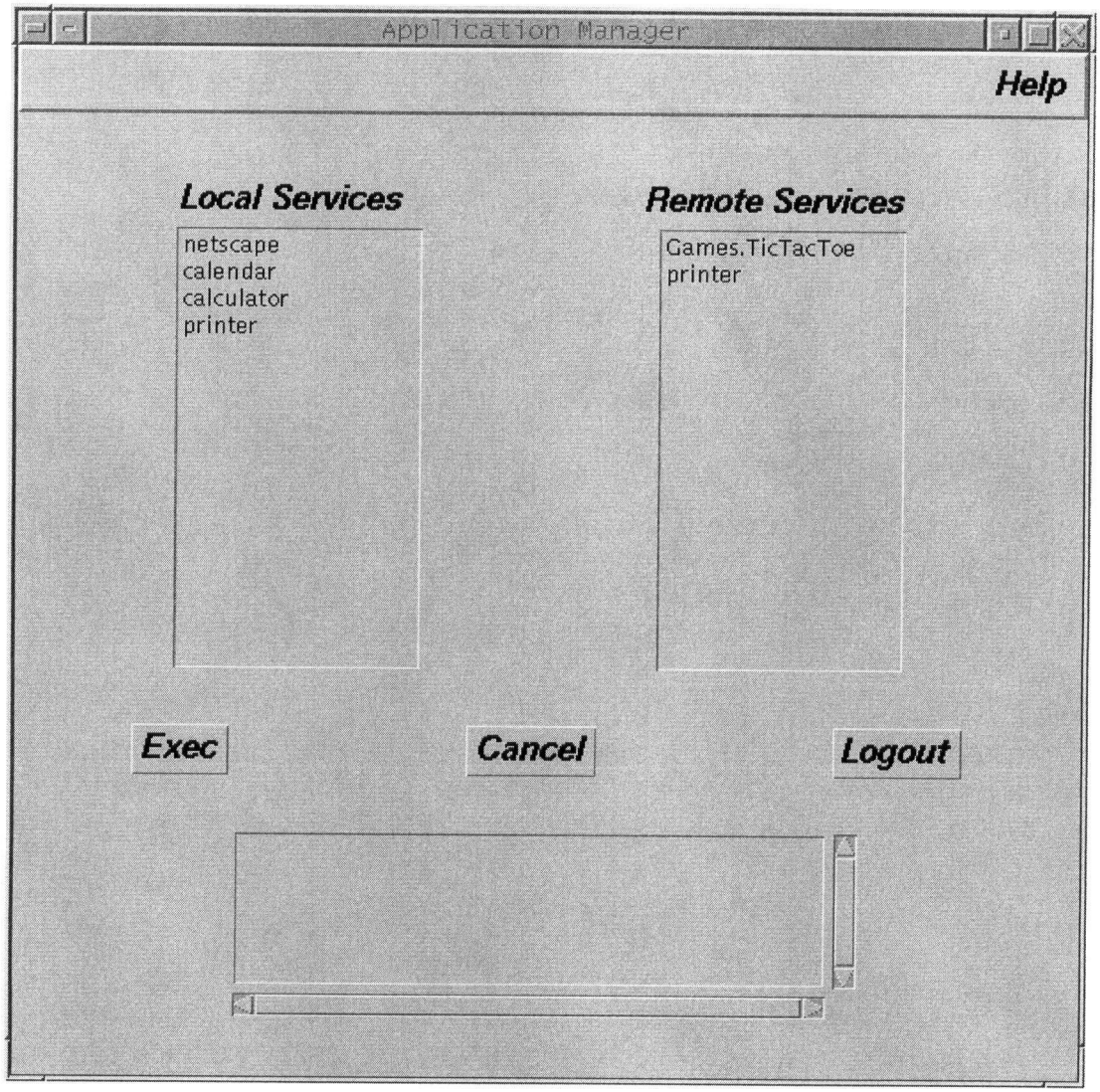

Figure 4.3: UPC Application Manager screen

the user selection, one of the following actions will be carried out.

- If the user chooses to run a remote application, the IA invokes the `runRemoteApp()` method from the TA with the appropriate parameters.
- If the user chooses to run a local application, the IA invokes the `runLocalApp()` from the TA with the appropriate parameters.

4.1.6 Testing and Results

Testing Preparation

The prototype supports mobile users with transparent access to their personalized computing environments wherever they roam on the Internet, using wireless or wired connections. To demonstrate this capability of the prototype, we have prepared the followings items:

- Software Application

  There are two main categories of network resources and software applications, i.e. local and remote. The "local" category includes all the network services that are available at the visiting network and all the software applications that are installed in the visited terminal. On the other hand, the networks services that are available outside the visiting network (i.e. at the home network) and software applications that can be remotely executed (i.e. applications developed based on CORBA's framework) fall into the "remote" category. For local application, we used the web browser (i.e. Netscape) as an example. The user would be able to run Netscape with the preference and configuration files (i.e. bookmarks, preference, cookie files) from their home network. In regards

to the remote application, we implemented a CORBA-based software application, a distributed TicTacToe game. It consists of a very thin application client (i.e. GUI) and an application server. A user can play a game half way and save it, then resume the previously paused game any time later.

- User

  We created a user called "test" under the administrative domain "ece.ubc.ca". In the user PCE, we specified the user would like to use netscape as their web browser, and the "bookmarks.html" and the "preferences" files are defined as the preference files for the application "netscape". In addition to this, the user wants to be able to play a TicTacToe game away from home.

- Terminal

  We configured a foreign terminal in a second administrative domain, "cs.ubc.ca", to support UPC, and created a terminal profile for this terminal using the Service Manager utility application. The terminal profile specifies that a web browser called "netscape" is installed at the terminal. The mapping information for all the reference and configuration files for netscape are also added to the terminal profile. For example "netscape" is installed at "/usr/bin/netscape" and the bookmarks file is mapped into the "/usr/bin/netscape/bookmarks.html" file.

Results

When the user "test" moves away from home and visits the "cs.ubc.ca" network, they are able to login and have the same computing environment as they have at home (i.e. ece.ubc.ca). The user can run netscape with the same bookmarks and

preferences as they have at home, and any change to the bookmarks or preferences files is synchronized with the corresponding files at the home network. The user can also remotely play a TicTacToe game and resumes a previously paused game.

To get a reading on the performance of the CORBA-based UPC prototype implementation, we have recorded the time that it takes to perform a task in different situations. The following shows some figures calculated based on an average of 80 runs:

- Time to resolve the User Agent : 702 msecs
- Time to process a login request : 174 msecs
- Download file rate : 27 bytes/msecs

Chapter 5

Conclusion and Future Works

5.1 Conclusion

This thesis addresses implementation issues regarding the support of mobile computing over the Internet using a new computing paradigm called Universal Personal Computing (UPC)[30,35]. UPC is a computing environment which enables a mobile user to access computing resources, network services, personal applications, data files, and environmental configurations through any terminal, stationary or mobile, anywhere on the Internet [21]. The core of UPC's mobility management is the Registration and Service Negotiation protocol (RSNP). We first specified the UPC's RSNP using Specification and Description Language (SDL). We then simulated RSNP using different scenarios to demonstrate the protocol's behaviour, and to verify the protocol's completeness and correctness. With the concrete and precise understanding gained from UPC's RSNP SDL specification, a prototype of UPC's RSNP was subsequently implemented in C and on Sun Solaris with the user interface written in Tcl/Tk.

The specification and simulation of RSNP using SDL has made the implementation of the UPC's RSNP prototype easier and simpler. In SDL specification, an agent's behaviour is described using an extended finite state machine (FSM) and coding from a FSM is a straightforward task. More important, the simulation has verified the response of a FSM against different combinations of inputs and states. The FSM is proved to correctly work before any coding attempt. This results in a huge reduction in debugging time in the implementation phase.

Unfortunately, using C language to implement all the distributive components in UPC is very complex and even impossible. The UPC prototype implemented using C language supports only one feature proposed in the UPC framework, i.e. executing a local application using the downloaded configuration and preference file from the user's home network. However, there is no support for other features such as remotely executing an application or a service in the home network from the foreign network, or downloading an application from the home network and executing at the foreign network. To successfully implementing all the features in the UPC framework, a combination of a platform independent distributed object architecture support (i.e. CORBA) and a platform independent programming language (i.e. JAVA) is necessary. The distributed object architecture will make the implementation of a distributed application easier than using only C language. Looking for an object, marshalling data from one language to another, or invoking a method on a remote object are performed transparently by the distributed object framework. In addition, the platform independence of Java will make all the client side of all the distributed components portable across platform.

As a result, this thesis has presented an alternative approach to modeling the UPC concept using distributed objects, called CORBA based UPC [30, 35]. In this

architecture, the UPC components are modeled as distributed objects with CORBA as the object bus that facilitates the interaction, integration, and distribution of these objects. To demonstrate the concept, a prototype of CORBA-based UPC was implemented using Java 1.1.x and Visigenic's CORBA for Java 3.0 on a Solaris platform.

With the advantage of using Java and CORBA, we were able to implement most of the features proposed in the UPC framework. The implementation is very simpler as compared to the C/Tcl/Tk implementation. The prototype implementation supports a mobile user to transparently access their configuration and preference file, and execute their personal software application whenever and wherever they roam on the Internet. The application could be a local application running with the user's configuration and preference file, or an application that is remotely executed at the user home network.

5.2 Future Works

Since the implementation of the CORBA-based UPC is only a prototype and some of the CORBA service implementations are not available at the time the prototype was implemented, many features remain to be implemented in the future.

- In the PCE, the set of values that describe the required hardware capabilities for a service remains to be standardized and implemented.
- In the terminal profile, the set of values that describes the terminal's hardware capabilities remains to be standardized and implemented.
- As the CORBA's Trader Service becomes available, the Terminal Agent could be enhanced to support service discovery at the visiting network.

- As the CORBA's Concurrency Control is available, the User Agent could use this service to control access to the PCE, and the Terminal Agent could use this service to synchronize file downloading and uploading.
- Porting this project into a Web-based framework would be a possible extension.
- Last but not least, a detailed study of the UPC's protocol performance and security issues needs to be conducted.

Bibliography

[1] R. Bagrodia, W.W. Chu, L. Kleinrock, and G. Popek, "Vision, Issues, and Architecture for Nomadic Computing", *IEEE Personal Communications Magazine,* pages 14-27, December 1995.

[2] F. Belina, D. Hogrefe, and A. Sarma, "SDL with Application from Protocol Specification", *Prentice Hall International (UK) Ltd.,* 1991.

[3] J. C. Benard-Dende, R. Nevoux, and J. C. Dang, "Networks, Users and Terminals in UMTS/FPLMTS", *Vehicular Technology Conference 44th.,* pages 681-685, 1994.

[4] R. Braek, "SDL Basics", *Handouts.*

[5] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik, " Itinerant Agents for Mobile Computing", *IEEE Personal Communications,* pages 34-49, October 1995.

[6] CORBA Home Page, *URL: http://www.omg.org,* 1998.

[7] T.V. Do, and J. Audestad, "Terminal Mobility Support in TINA", *Proceedings of TINA's 97,* pages 38-50, 1998.

[8] T. Eckardt, and T. Magedanz, "The Role of Personal Communications in Distributed Office Environments", *Autonomous Decentralized Systems Proceedings,* pages 316-322, 1995.

[9] T. Eckardt, and T. Magedanz, "Personal Communications Support based on TMN and TINA Concepts",*Proceedings of IN's 96,* pages 196-200, 1996.

[10] M. P. Gervais, "A Framework for Mobility in Wireless Personal Communications", *Proceedings of ICC/SUPERCOMM's 96,* pages 1148-1152, vol. 2, 1996.

[11] V. Gupta, and A. Dixit, "The Design and Deployment of a Mobility Support Network",*Proceedings Second International Symposium on Parallel Architectures, Algorithms, and Networks,* pages 228-234, 1996.

[12] J. G. Hemmady, J. R. Maymir, and D. J. Meyers, "Network Evolution to Support Personal Communications Services", *Global Telecommunications Conference,* pages 710-714, vol. 21994.

[13] T. D. Hodes, and R. H. Katz, "Composable Ad hoc Location-based Services for Heterogeneous Mobile Clients", , 1997.

[14] C. S. Hong, Y. Koga, and Y. Matsushita, "A Networking Architecture for Mobility Services Using Mobile Agent Approach", *Proceedings of TINA's 97,* pages 297-307, 1998.

[15] I. Iida, T. Nishigaya, and K. Murakami, " DUET: An Agent-Based Personal Communications Network", *IEEE Communications Magazine,* pages 44-49, November 1995.

[16] T. Imielinski, and H. F. Korth, "Mobile Computing", *Kluwer Academic Publishers* , 1996.

[17] J. Ioannidis, and G. Q. Maguire Jr., "The Design and Implementation of a Mobile Internetworking Architecture", *Winter USENIX,* 1993.

[18] E. Jung, Y. J. Park, and C. Park, "Mobile Agent Network for Supporting Personal Mobility", *Proceedings Twelfth International on Information Networking,* pages 131-136, 1998.

[19] M. Liljeberg, K. Raatikainen, M. Evans, S. Furnell, N. Maumon, E. Veldkamp, B. Wind, and S. Trigila, "Using CORBA to Support Terminal Mobility", *Proceedings of TINA's 97,* pages 59-67, 1998.

[20] Y. Li, and V.C.M. Leung, "Supporting Personal Mobility for Nomadic Computing over the Internet", *ACM Mobile Computing and Communications Review* , Vol. 1, No. 1, pages 22-31, April 1997.

[21] Y. Li, and V.C.M. Leung, "Protocol Architecture for Universal Personal Computing", *IEEE Journal on Selected Areas in Communications* , Vol. 15, No.8, pages 1467-1476, October 1997.

[22] A. Lombardo, P. Nicosia, S. Palazzo, and M. Samarotto, "Service Architecture Support of Personal Mobility in a Multi-Domain Environment", *Proceedings of TINA's 97,* pages 51-58, 1998.

[23] R. Orfali, and D. Harkey, "Client/Server Programming with JAVA and CORBA", *John Wiley and Sons, Inc.,* 1997.

[24] J. Pavon, and J. Tomas, "CORBA for Network and Service Management in the TINA Framework", *IEEE Communications Magazine,* pages 72-79, March 1998.

[25] C. Perkins, Ed., RFC2002, IP Mobility Support, March 1997.

[26] R. Ramjee, T.F. LaPorta, and M. Veeraragahvan, "The Use of Network-Based Migrating User Agents for Personal Communication Services", *IEEE Personal Communications,* pages 62-68, December 1995.

[27] D. Samfat, and R. Molva, "A Method Providing Identity Privacy to Mobile Users during Authentication", pages 196-199, *Workshop on Mobile Computing Systems and Applications,* pages 196-199, 1995.

[28] A. Sarma, "Introduction to SDL-92", *Handouts.*

[29] H. M. Sneed, " Encapsulating Legacy Software for Use in Client/Server Systems", *Proceedings of WCRE'96,* pages 104-119, 1996.

[30] M. Toeroe, J. Zhu, Y. Li, and V.C.M Leung, "A CORBA based Framework for Univeral Personal Computing on the Internet",*Proceedings SCI/ISAS'98, Orlando, FL,* July 1998.

[31] Vinoski S., "CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments, *IEEE Communications Magazine,* pages 46-55, vol. 35, 1997.

[32] S. Vinoski, "Distributed Object Computing With CORBA", *C++ Report Magazine,* July/August 1993.

[33] Visigenic Programmer's Guide, Version 3.0, VisiBroker for Java.

[34] M. Zaid, "Personal Mobility in PCS", *IEEE Personal Communications Magazine* , pages 12-16, Fourth Quarter 1994.

[35] J. Zhu, M. Toeroe, V.C.M. Leung, and S. Vuong, "Supporting Universal Personal Computing on Internet with Java and CORBA", *Concurrency: Practice and Experience,* 1998.

Appendix A

The IDL files for all the agents in the CORBA-based UPC architecture.

```
//Author : Thong Huynh
module TA {
     interface TerminalAgent{
          enum FailReason { UNABLE_TO_LOCATE_SERVER,
                            NO_IMPLEMENT_SERVER,
                            CLASS_NOT_FOUND,
                            INSTANTIATION_ERROR,
                            ACCESS_ERROR,
                            CLASSFILE_DOWNLOAD_ERROR,
                            ERROR_EXEC_APPLICATION,
                            UNKNOWN_HOST};
          exception Fail{
               FailReason reason;
          };
          void updateLocation(in string newLocation) raises (Fail);
          boolean login(in string LUI, in string password) raises (Fail);
          void logout() raises (Fail);
          string runLocalApp(in string appName) raises (Fail);
          string runRemoteApp(in string appName) raises (Fail);
          string getLocalServiceList() raises (Fail);
```

```
            string getRemoteServiceList() raises (Fail);
        };
};


//Author : Kangming Liu
module PCE {
        typedef string ServiceName;
        typedef string ServiceObj;
        typedef string ServiceTypeName;
        typedef string Constraint;
        typedef string Preference;
        typedef string PolicyName;
        typedef string PropertyName;
        typedef string PolicyValue;
        typedef string PropertyValue;
        typedef string HpropertyValue;
        enum PropertyMode {
            PROP_NORMAL,
            PROP_READONLY,
            PROP_MANDOTORY,
            PROP_MANDOTORY_READONLY
        };
```

```
enum ObjectRefType {
    APPLI_OBJECT,
    APPLI_CONFIG,
    APPLI_RESOURCE_FILE
};
struct HardwareCap{
    string Hname;
    HpropertyValue value;
};
typedef sequence<HardwareCap>HardwareCapSeq;
struct Property {
    PropertyName name;
    PropertyValue value;
    PropertyMode mode;
};
typedef sequence <Property> PropertySeq;
struct Policy{
    PolicyName name;
    PolicyValue value;
};
typedef sequence<Policy>PolicySeq;
struct Trade{
    ServiceTypeName type;
    Constraint constr;
    Preference pref;
```

```
        PolicySeq policies;
        PropertySeq Properties;
    };
    struct Service{
        ObjectRefType type;
        ServiceName serviceName;
        ServiceObj serviceObj;
        Trade trade;
        HardwareCapSeq hardwareCapSeq;
    };
    typedef sequence<Service>ServiceSeq;
    interface Profile{
        attribute ServiceSeq serviceSeq;
        void set(in PCE::ServiceSeq serviceSeq);
        PCE::ServiceSeq get(in string username);
    };
    interface ProfileFactory{
        Profile createProfile(in string User_name,
                              in ServiceSeq serviceSeq);
        void deleteProfile(in Profile p);
    };
};

module UA {
    struct Token{
```

```
        string low;
        string high;
    };
    interface UserAgent {
        boolean authenticate(in string LUI,
                             in string password);
        boolean tokenauthent(in string LUI,
                             in string password,
                             out Token token);
        void inform_location(in string LUI,
                             in string address);
        PCE::Profile getprofile(in string username);
        string download(in string filename);
        boolean upload(in Token token,
                       in string filename,
                       in string data);
        boolean updateProfile(in string username,
                              in PCE::Profile p);
        void logout(in string username);
    };
};
```