GLOBAL CONTEXT INFORMATION TRANSFER: ONE WAY TO ACHIEVE UBIQUITOUS SERVICE ACCESS

by

JIEXIN LUO, M.Sc.

A thesis submitted to
the faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science

Carleton University
Ottawa, Ontario
May 8, 2003

© copyright
2003, Jiexin Luo

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-83491-3

Our file Notre référence

ISBN: 0-612-83491-3

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

Canada

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

**GLOBAL CONTEXT INFORMATION TRANSFER: ONE
WAY TO ACHIEVE UBIQUITOUS SERVICE ACCESS**

submitted by Jiexin Luo, M.Sc.

in partial fulfillment of the requirements for
the degree of Master of Computer Science

A handwritten signature in dark ink, appearing to read "Bernard Pagé", written over a horizontal line.

Thesis Supervisor

A handwritten signature in dark ink, appearing to read "Frank Deh", written over a horizontal line.

Director, School of Computer Science

May 8, 2003

Carleton University

Abstract

Real-time / business-critical web applications are built on “value-adding” network protocols. The operations of these protocols require refined configuration data and state variables. Currently, most of the network protocols get configured via static means. However, this approach fails to accommodate mobile networks. In order to bring mobility support, the Seamoby working group proposed context information transfer, i.e. transferring protocol configuration data and state variables from one network element to another during handover. Seamoby’s proposal focuses mainly on local area networks.

Enlightened by the research from the Seamoby working group, this project proposed global context information transfer to set up network services when mobile users are in foreign networks. This project has firstly designed and prototyped the Context Information Publishing Framework to help achieve an unambiguous understanding of context information. Secondly, this project has provided an end-to-end solution for transferring context information securely across the Internet.

Acknowledgements

I would like to thank my supervisor Dr. Bernard Pagurek for his intelligence, enthusiasm, and guidance through the course of this project. I would also like to thank my parents and my son for their support and understanding during the two years.

Special thanks to Mr. Vladimir Tasic for conveying valuable knowledge on Web Services and related subjects.

This project was supported by Communication Information and Technology Ontario (CITO).

Table of contents

Table of contents-----	3
List of figures-----	6
Glossary -----	9
Chapter 1 Introduction-----	12
1.1 Motivation-----	12
1.2 Major contributions -----	13
1.4 Related works-----	14
1.3 Thesis organization -----	15
Chapter 2 Wireless IP networks-----	16
2.1 Introduction -----	16
2.2 Data networking solution from 3GPP-----	17
2.3 Data networking solution from 3GPP2 -----	20
2.4 Mobile IP -----	23
2.5 Summary -----	24
Chapter 3 QoS and security management in IP networks-----	26
3.1 Introduction -----	26
3.2 QoS management in IP networks -----	26
3.2.1 The IntServ architecture -----	27
3.2.1.1 QoS services provided by IntServ -----	27
3.2.1.2 How packets are served in IntServ nodes-----	28
3.2.2. The DiffServ architecture -----	29
3.2.2.1 QoS services provided by DiffServ-----	31
3.2.2.2 How packets are served in DiffServ nodes -----	32
3.3 The IPSec protocol-----	36
3.4 Summary-----	39
Chapter 4 Context Information Transfer-----	41

4. 1 Problem statement: lack of mobility support in QoS, security management protocols	41
4.2 Context information transfer proposal from the Seamoby working group	45
4.2.1 Context information definition	45
4.2.2 Local context information transfer	49
4.3 Global context information transfer	49
4.3.1 Definition of global context information transfer	49
4.3.2 Three ways to arrange global context information transfer	53
4.4 Global context information transfer and the Virtual Home Environment	54
4.5 Summary	56
Chapter 5 Context Information Publishing Framework	58
5.1 Overview	58
5.2 Defining context information in CIPF	61
5.2.1 Defining context information	61
5.3 Publishing context information	69
5.3.1 Review of UDDI technology	69
5.3.2 Publishing tModel entities in UDDI registry	75
5.4 The CIPF application	77
5.4.1 The purpose of the CIPF application	77
5.4.2 Use case description for the CIPF application	78
5.4.3 Design and prototype of the CIPF application	81
5.4.3.1 Overview	81
5.4.3.2 XML digital signature technology and XML key management technology	83
5.4.3.3 Design and prototype of the presentation tier	87
5.4.3.4 Design and prototype of the business tier	94
5.4.3.5 Development tools	97
5. 5 Summary	97
Chapter 6 Global Context Information Transfer	100
6.1 Overview	100
6.2 Service Discovery phase	103

6.2.1 Overview of the Service Location Protocol -----	103
6.2.2 Prototyping of SLP user agent by this project -----	105
6.3 Context information transfer-initiating phase -----	109
6.3.1 Message description -----	109
6.3.2 Functional description -----	112
6.3.2.1 Functional description for the home Context Information Host -----	112
6.3.2.2 Functional description for the Mobile device -----	113
6.3.3 Prototyping by this project -----	114
6.3.3.1 Implementation in the home Context Information Host -----	114
6.3.3.2 Implementation in mobile devices -----	117
6. 4 Context information transfer phase -----	120
6.4.1 The Secure Socket Layer protocol -----	122
6.4.1.1 Review of the Secure Socket Layer protocol -----	122
6.4.1.2 Open source code and library -----	125
6.4.2. The Context Information Protocol -----	126
6.4.2.1 Message description -----	126
6.4.2.2 Functional description -----	130
6.4.2.3 Prototyping in this project -----	132
6.4.3 The CT-Security Layer -----	139
6.4.3.1 Message description -----	139
6.4.3.2 Functional description -----	143
6.4.3.3 Prototyping -----	146
6.5 Mobile device starting service-centric applications -----	150
6.6 Testing of prototype implementation -----	150
6.7 Summary -----	156
Chapter 7 Summary and future works -----	158
7.1 Summary of this project -----	158
7.2 Conclusions -----	159
7.2 Future work -----	160
References -----	162

List of figures

Figure 2.1(a): GPRS architecture	18
Figure 2.1(b): Structure of one GPRS subnet	18
Figure 2.2: Data accessing in all_IP 3GPP networks	19
Figure 2.3: 3GPP2 network architecture	21
Figure 2.4: Protocol stack for MN, BSS and PDSN	21
Figure 3.1: The structure for a DS domain	31
Figure 3.2: Components for a DS boundary node	33
Figure 3.3: The basic operations of IPSec	37
Figure 4.1: Business-critical applications such as the videoconference application described in our scenario can be served by virtual private networks	42
Figure 4.2: Sketch for Ottawa and Toronto public networks mentioned in the example scenario	44
Figure 4.3: The Internet is divided into multiple CIT domains	52
Figure 4.4(a): Current network protocols and network applications	55
Figure 4.4(b): Context Information Transfer can be added at application level	55
Figure 5.1: The three types of entities for defining information	58
Figure 5.2: The XML schema definition for traffic profile.	60
Figure 5.3: Information management in CIPF	61
Figure 5.4: Relationship between Transfer Data Definition Schema, Transfer Data Definition Document and content of context information messages	64
Figure 5.5(a): The structure of a Transfer Data Document	66
Figure 5.5(b) Transfer Data Definition Schema	68
Figure 5.6: An example of a Transfer Data Definition Document	69
Figure 5.7: tModel entity X is derived from tModel entity Y	73
Figure 5.8: The XML schema definition for the DiffServe EF service level	76
Figure 5.9: The tModel element for the DiffServ EF service level definition	76
Figure 5.10: Design for the CIPF application	82
Figure 5.11: The CIPF application delegates part of the digital signature processing to dedicated trust service server	86

Figure 5.12: The design of the presentation tier in the Context Publish Framework	89
Figure 5.13: The GUI for publishing tModel entities for Protocol Configuration Schemas	90
Figure 5.14: The GUI for publishing tModel entities for Transfer Data Definition documents	91
Figure 5.15: The Storage bean component	96
Figure 6.1: The three phases in Global Context Information Transfer	100
Figure 6.2: The class diagram for implementing SLP user agent	106
Figure 6.3: WTK20 directory layout	108
Figure 6.4: The format for Transfer initiating message	109
Figure 6.5: The plaintext format for of the encrypted payload in the Transfer-initiating	110
Figure 6.6: The format for Transfer-Initiating Acknowledgement Message	111
Figure 6.7: The plaintext content for of the encrypted payload in the Transfer Initiating Acknowledgement message	111
Figure 6.8: Class diagram for the classes implemented in the home Context Information Host	115
Figure 6.9: Two hash maps are used to hold secret keys for all mobile devices	115
Figure 6.10: Class Encryptor uses packages developed by Bouncy Castle	118
Figure 6.11: Two options to locate Context Information Protocol	120
Figure 6.12: Message sequence diagram for SSL protocol	123
Figure 6.13: The messages exchanged in the Context Information Protocol	126
Figure 6.14: The format for the messages exchanged in the Context Information Protocol	127
Figure 6.15: The payload for the Meta-data message	128
Figure 6.16: State diagram for the home Context Information Host	131
Figure 6.17: The state diagram for the foreign Context Information Host	132
Figure 6.18: Classes diagram for the sender in global context information transfer	133
Figure 6.19: The pseudo code for the method send() in Sender class	134
Figure 6.20: Classes to be installed in the foreign Context Information Host	137
Figure 6.21: The messages exchanged to set up the CT-Security Layer connection	140
Figure 6.22: Format for messages passing through CT-Security layer	140
Figure 6.23: Payload for the Client Public Key message	141

Figure 6.24: Message format for the Server Public Key Message	142
Figure 6.25: The state diagram for the home Context Information Host in the CT-Security Layer	143
Figure 6.26: The state diagram for the foreign Context Information Host	144
Figure 6.27: Class diagram for the CT_Security Layer classes installed in the home Context Information Host	145
Figure 6.28: Classes implemented in the foreign Context Information Host for the CT-Security Layer	149
Figure 6.29: Testing of the prototype classes	151
Figure 6.30(a): The UML message sequence diagram describing the interactions among M application, F application, H application and SLP dummy server. H application and F application communicates via CT-Security layer	155
Figure 6.30(b): The UML message sequence diagram describing the interactions among M application, F application, H application and SLP dummy server. H application and F application communicates via SSL	156

Glossary

CIT, Context information Transfer: The IETF Seamoby working group defines context information transfer as “the movement of context from one router or other network entity to another as a means of re-establishing specific services on a new subnet or collection of subnets”. It is renamed as local context information transfer in this project

CIT domain: A concept proposed by this project. One CIT domain is the basic unit for local context information transfer

Context information: A concept defined by the IETF Seamoby working group. Context information is information “on the current state of a service required to re-establish the service on a new subnet without having to perform the entire protocol exchange”.

Context information host: The application gateways that can communicate context information

Context Information Protocol: An application level protocol for communicating context information. It is defined by this project

Context Information Publish Framework (CIPF): The framework proposed by this project for defining and publishing context information. The purpose of CIPF is to achieve semantic understanding of context information by developers in a global scope

CIPF application: The web application designed to facilitate the use of CIPF

CT-Security Layer: An application level security protocol developed by this project. CT-Security Layer comes into play when an SSL connection is not possible

DiffServ: The scalable QoS architecture proposed by the IETF

Digital Encryption Standard (DES): A private key encryption algorithm

Digital Signature Algorithm (DSA): A popular public key algorithm

Enterprise JavaBeans (EJB): The architecture for component-based, distributed computing in the Java platform

Foreign context information host: The Context Information Host located in the foreign network for a mobile device

Global context information transfer: A concept proposed by this project. Global context information transfer is defined as the “action of passing context information from

one network element to another remote network element, via the public networks. The sender and the receiver may belong to different management authorities and not know each other beforehand. The purpose of global context information transfer is to setup network services”

GPRS: Stands for General Packet Radio Service (GPRS). GPRS is the architecture providing packet-switched data services in GSM

GSM: Stands for the Global System for Mobile Communication, a set of wireless cellular network standards developed in Europe

Home context information host: The Context Information Host located in the home network for a mobile device

IntServ: An IP per-flow-based QoS management architecture proposed by the IETF

IPSec: An IP layer security protocol proposed by the IETF

Java™ 2 Micro Edition (J2ME): It is the Java™ 2 platform targeted at consumer electronics and embedded devices

JavaServer Pages (JSP): A Java web technology for composing dynamic web content

Mobile IP: The IETF standard to add mobility support in the IP layer

Quality of Service (QoS): The service quality provided by the network traffic flows

Public Key Infrastructure (PKI): The set of infrastructure services that support the wide-scale use of public key cryptography

Robust Header Compression protocol (ROHC): A robust header compression protocol developed by the IETF

RSA: Public key encryption / digital signature algorithm invented by Ron Rivest, Adi Shamir and Len Adleman

Security Associate (SA): Defined by IPSec standard as a simplex ‘connection’ that affords security services to the traffic carried by it

Security Association Database (SAD): Defined by IPSec as nominal database which contains parameters for the afforded security services in a IPSec enabled node

Service-centric application: A service-centric application is normally network-based, and the forwarding of its traffic requires quality of service (QoS) and security management. A service-centric application normally has real-time / business-critical nature

Service Location Protocol: An IETF protocol for discovering local network services

Security Parameter Index (SPI): SPI is a 32-bit value used in the operation of IPSec for distinguishing packets with the same destination address and the same IPSec protocol

Secure Socket Layer: Originally developed by the Netscape cooperation, it has become the *de facto* standard for transport layer security

tModel entity: tModel” is the abbreviation for Technical Model, and it is the basic mechanism in UDDI to represent “unique concepts or constructs” including specifications, protocols or namespaces

Traffic flow: One traffic flow is identified by an IP source address, an IP destination address and port number. It is also named as IP flow in this project

UDDI: Stands for Universal Description, Discovery and Integration. UDDI is the public registry for the emerging Web Service architecture

Virtual Home Environment (VHE): Defined by 3GPP as a concept for personal service environment portability across network boundaries and between terminals. The concept of VHE is such that users are consistently presented with the same personalized features, user interface customization and services in whatever network and whatever terminal, wherever the user may be located

XML: Stands for eXtensible Markup Language. The *de facto* standard data format used in web communications

XML digital signature standard: The XML signature standard provides a uniformed way to communicate digital signature values as well as information required to verify the signature. It is developed by W3C

XML key management standard: A specification of XML application/protocol that allows a simple client to obtain key information (values, certificates, management or trust data) from a web service. It is developed by W3C

XML schema: An XML schema is the new way to specify the legal structure for any XML document

Chapter 1 Introduction

1.1 Motivation

The goal of this project is to investigate global context information transfer as a mechanism for facilitating the running of service-centric applications on the mobile Internet. A service-centric application is normally web-based, and the forwarding of its traffic requires quality of service (QoS) and security management. A service-centric application normally has a real-time / business-critical nature. An example of service-centric applications is the web videoconference application described in the following motivating scenario:

Imagine that we have a user in Ottawa who is scheduled to attend a two-hour long Internet videoconference with his Ohio colleagues every Monday morning. The user usually joins the conference from his office in Ottawa. All traffic for the videoconference has to be encrypted by the IPsec protocol (reviewed in Chapter 3) and tunneled through his company's Virtual Private Network (VPN), since critical business data is involved. The real-time nature of the videoconference application requires that its traffic should be served with higher priority. Assume that the DiffServ architecture (reviewed in Chapter 3) is deployed, and Expedited Forwarding Per Hop Behavior (EF PHB) is used to serve the videoconferencing application. EF PHB is a DiffServ service level for achieving low delay and low jitter. Suppose that the company's network in Ottawa is configured appropriately so that the videoconference is set up and runs smoothly every week. But on one Monday morning, although the user is on a business trip to Toronto, he cannot afford

to miss the conference. Our user, therefore, decides to join the conference from Toronto airport using his powerful handheld device and to continue to participate while he walks to the hotel, where he will stay and finish the conference.

The high bandwidth capacity of 3G cellular networks, the IP level mobility management, and the Internet QoS security management protocols lay the foundation for achieving the above scenario. However, as described in chapter 4, we still need a global mechanism to communicate the traffic processing requirements so that mobile users can run service-centric applications on visiting networks. Consider our scenario. When the Ottawa user travels to Toronto and begins the videoconference from Toronto airport, how can the local network be configured quickly so that proper QoS / security services are applied to the IP traffic of the videoconference application? The goal of this thesis is to investigate how to provide a solution to the above challenge using global context information transfer.

1.2 Major contributions

Briefly, context information is network protocol configuration information required to serve a specific traffic flow. Context information transfer is the action of passing context information from one network element to another when a mobile device changes its location (detailed discussion in Chapter 4). Originally, context information transfer was proposed by the IETF Seamless Mobility working group as a means to decrease the disruptive effect of handovers in a local network. The first contribution of this project was to extend context information transfer to the Internet scope, and propose

global context information transfer as one way to enable ubiquitous service access across the Internet.

Global context information transfer occurs between a mobile device's home network and a visited foreign network, which may belong to different management authorities. How to make the context information sent by the home network understandable to the foreign network is a major challenge to be overcome in order to achieve global context information transfer. This project contributes by designing the Context Information Publishing Framework (CIPF). CIPF specifies guidelines on how to define and publish context information in a co-operative way. This project has designed and implemented the *Transfer Data Definition Schema*, an XML schema document that acts as the template for specifying what information is to be transferred as one unit when starting a service-centric application. In order to facilitate information publishing, this project has also prototyped the CIPF application.

Finally, this project has shown in detail how global context information transfer can be integrated with the current Internet architecture. One end-to-end solution, based on well-established network protocols and a portable mobile device platform, has been proposed and prototyped.

1.4 Related works

This project was derived from the context information transfer proposal that was originally studied by the IETF Seamless Mobility working group.

1.3 Thesis organization

The rest of the thesis is organized as follows:

A review of how the IP technology and wireless network have been integrated is given in Chapter 2.

A review of several “value-adding” data network protocols, including the Integrated Service Protocol, the Differentiated Service Protocol and the IP Security protocol, is presented in Chapter 3.

Chapter 4 introduces the concept of context information and global context information transfer. A blueprint for building an infrastructure to support communicating context information globally is also given in Chapter 4.

Chapter 5 describes the Context Information Publishing Framework, which includes a set of guidelines for defining and publishing context information, and a Web application designed for facilitating an unambiguous understanding of context information in the global scope.

Chapter 6 illustrates how context information can be communicated securely across the Internet using well-established network protocols and technologies.

Chapter 7 summarizes the project and provides some perspectives on future work.

Chapter 2 Wireless IP networks

2.1 Introduction

Wireless networking is an important driving force for today's high-tech industry. The term "wireless networking" is almost a synonym for Personal Communication Service (PCS), which "refers to a wide variety of wireless access and personal mobility services provided through a small terminal, with the goal of communications at any time, at any place and in any form" [1]. Over the past thirty years, a genre of technologies has been developed in this area; examples include cellular mobile phone systems, public and residential cordless access systems, wireless local area networks, and wireless personal area networks. Of the above technologies, the most popular and successful is cellular mobile telephony. There are three generations of cellular wireless telephony systems. The first generation cellular systems (1G) made use of analog technology. Advanced Mobile Phone Service (AMPS) was a typical 1G system and achieved great commercial success; however, 1G was replaced by digital cellular systems, i.e. the second generation mobile phone systems (2G). Compared with analog technology, the digital alternatives provide higher user densities and lower operation costs. One of the most influential 2G systems is GSM, the Global System for Mobile Communication, a set of wireless cellular network standards developed in Europe. GSM, like other 2G systems, is continuously evolving, with the concentration of this evolution on adopting more advanced radio access technologies and integrating the packet-switched data networks, i.e. the Internet, to cellular systems. Providing advanced data services in addition to traditional circuit-

switched telephony is the goal of cellular network operators. GSM phase 2+ (2.5 G, as it is called by some authors) already has the capacity to support Internet applications [1]. The third generation systems (3G) are expected to support both high-speed wireless Internet access (up to 2 Mbps) and wireless multimedia services [2]. Data networks have become an inseparable part of cellular systems.

This chapter aims to provide a review of how cellular mobile systems are incorporated with packet-switched networks. Two architectures are discussed in the rest of the chapter. The architecture proposed by 3GPP [3][4] is reviewed in section 2.2, while the one proposed by 3GPP2 [5] is discussed in section 2.3. Finally, Mobile IP, the IETF's effort for adding mobility management in the IP layer, is briefly summarized in section 2.4.

2.2 Data networking solution from 3GPP

3GPP stands for the "Third Generation Partnership Project". It is a concerted effort in Europe which aims to produce a set of standards for 3G mobile systems, with GSM as the core cellular network technology.

3GPP started with the General Packet Radio Service (GPRS) to provide packet-switched data services. Initially, GPRS is supposed to co-exist with GSM circuit-switched voice services. The architecture of a GPRS network comprises a set of GPRS supporting nodes (GSN). There are two types of GSN: the supporting GSN (SGSN) and gateway GSN (GGSN). The GPRS architecture also includes backbone networks that connect GGSNs as border gateways and handle packets between different GPRS

networks [3]. The GPRS architecture is shown in Figure 2.1(a), while the structure of one GPRS subnet is shown in Figure 2.1(b)

Figure 2. 1(a): GPRS architecture

Figure 2. 1(b): Structure of one GPRS subnet

GSNs are the gateways between GPRS networks and external packet data networks. SGSNs interface GPRS networks and radio access networks. As shown in Figure 2.1(b) above, SGSNs are connected with major GSM core network elements, including the Home Location Register (HLR), Equipment Identity Register (EIR) and Base Station (BSS). The location information of mobile terminals that are involved in packet data services is provided by GGSNs, which also tunnel packets received from external networks to appropriate SGSNs based on the destination addresses. SGSNs switch packets to correct BSS. Other responsibilities of SGSN include ciphering, authentication, session management, and mobility management [3].

Figure 2. 2: Data accessing in all_IP 3GPP networks

An all-IP 3GPP network architecture trend has appeared since mid-1999. To support an all-IP architecture, GPRS is overlaid with call control and gateway functions for supporting voice over IP (VoIP). The 3GPP all-IP reference architecture is shown in

Figure 2.2 (based on [4] [5]). CSCF (Call State Control Function) has analogous functions to the call control component in the circuit-switched technology.

A QoS profile is associated with every mobile terminal when attached to a GPRS network. The QoS profile contains information on traffic precedence, delay, reliability, peak throughput and mean throughput. SGSNs are responsible for fulfilling these QoS requirements. SGSNs can also modify a mobile node's QoS profile. After activating a certain QoS profile, the SGSN is responsible for ensuring data flow to the mobile terminal in accordance with its QoS profile. However, data flows within a core GPRS network are all served using the same best effort IP service. The QoS parameters are only used to reserve resources or to acquire differentiated service outside the GPRS core network. Inside a GPRS core network, communication between SGSN and GGSN is based on IP tunneling which is difficult to apply to the IP QoS framework. Different solutions have been proposed for service quality control in GPRS core networks [6].

2.3 Data networking solution from 3GPP2

The Third Generation Partnership Project2 (3GPP2) is a cooperation of national standard bodies. 3GPP2 partners include ARIB, TTC from Japan, CWTS from China, TTA from Korea and TTA from North America. 3GPP2 aims to develop a new packet-switched architecture building upon CDMA2000 air interface services. The 3GPP2 wireless network architecture is shown Figure 2.3 (adapted from [7]).

In 3GPP2's architecture, both simple IP and mobile IP are deployed in cellular systems to provide access to the data networks (the Internet or various intranets). A Packet Data Serving Node (PDSN), shown in Figure 2.3, is the entity that provides IP

functionality to the radio access network. “PDSN establishes, maintains, and terminates link layer sessions to the mobile station” [8]. A PDSN acts as a router if a simple IP is employed and may connect with several base stations. A PDSN plays the foreign agent’s role when being mapped to the Mobile IP architecture [8].

Figure 2.3: 3GPP2 network architecture

Figure 2. 4: Protocol stack for MN, BSS and PDSN

The protocol stacks for the mobile node (MN, as shown in Figure 2.3 and in Figure 2.4), base station (BSS, as shown in Figure 2.3 and in Figure 2.4) and PDSN are illustrated in Figure 2.4 (based on [9]).

3GPP2 selects the Point-to-Point protocol (PPP) as the data link protocol between mobile stations and PDSNs. A PPP session must be established before exchanging IP packets [9]. Choosing to terminate PPP in PDSN rather than in BSS is partly due to handover efficiency consideration. However, this design also leads to a scalability problem for PDSN and it is expected that PPP will be moved to BSS in future versions [10]. R-P is the interface between BSS and PDSN. Three options are available for implementing R-P. The three choices are:

1. Using existing wireless-specific interfaces from the standardization effort or from vendor implementations.
2. Using Layer-2 tunneling protocol for PPP developed by IETF.
3. Using mobile-IP based Layer-3 tunneling protocol developed by IETF. [9]

In addition to HLR (Home Location Register) / VLR (Visitor Location Register)-based authentication, PDSN is connected to an authentication, authorization and accounting (AAA) infrastructure.

Mobile stations may optionally support, and PDSNs are required to support, the IP DiffServ protocol [8]. If a mobile node has already marked packets with a certain DiffServ class, PDSN may accept the marking or re-mark them based on QoS requirement information acquired from the AAA servers. If packets are not marked by

the mobile mode, PDSN may optionally do the marking. PDSNs can also handle differentiated service for encapsulated IP packets [8].

2.4 Mobile IP

Mobile IP is the IETF's standard protocol for adding mobility support at the IP layer. It enables a wireless network node to move freely from one subnet to another, without disturbing the TCP end-to-end connectivity. Both Mobile IPv4 and Mobile IPv6 have been standardized; however, at present only Mobile IPv4 has been widely accepted. The following paragraph summarizes the basic principles of Mobile IPv4 [11].

In Mobile IPv4, a mobile node is given a home address which is a long-term IP address on a home network. The home network for a mobile node is defined in [11] as "a network, possibly virtual, having a networking prefix matching that of a mobile node's Home Address". All IP packets destined to a mobile node's home address are delivered to the mobile node's home network. Any network other than the mobile node's home network is named as a foreign network. Inside a mobile node's home network, there is at least one home agent, which is the router tunneling datagrams to the mobile node when it is in a foreign network.

When a mobile node enters a foreign network, it will first get a new, temporary IP address, i.e. the care-of-address, from the local foreign network. The care-of-address reflects the mobile node's location. After a care-of-address has been assigned, the mobile node registers the new care-of-address with its home agent, which keeps a table matching the home address with the care-of-address for every mobile node. The change of care-of-address is transparent to all of the parties that are communicating with the mobile node.

IP Packets destined to the mobile node are still labeled using its home address, and these packets are received by the mobile node's home agent, which tunnels them to the registered care-of-address of the mobile node [11].

2.5 Summary

Wireless networks and the Internet are the fastest growing areas in telecommunications. It is expected that convergence of these technologies is unavoidable, and the adoption of a full IP architecture in wireless communications has been proposed and studied by international standard organizations. With advanced radio access technologies and powerful mobile terminal equipment, fast wireless Internet access, wireless Voice over IP, and multimedia applications will be supported in 2.5 / 3G cellular networks. These IP-based services are expected to be very important for creating revenue in the future. However, there are still a number of open issues. For example, IP was originally designed for best effort services, while the majority of wireless traffics have a real-time nature. To guarantee satisfactory performance, IP QoS schemes have to be deployed, but both the Integrated service (IntServ) and the Differentiated service (DiffServ) have been designed without taking mobility into consideration. How to extend the IntServ protocol and the DiffServ protocol to mobile networks is going to be a great challenge.

In addition to QoS management, security is also a very important area in network communications. Most of the business-critical applications have to be protected by certain security means, such as digital signature and data encryption. The IPSec protocol is the new IETF security standard in the IP layer. Similar to QoS management protocols,

IPSec is also designed for fixed networks. How to extend the IPSec protocol to mobile networks is also going to be a great challenge.

Before discussing how to add mobility support to these protocols, I will first give a review on IP QoS and IP security solutions in the next chapter.

Chapter 3 QoS and security management in IP networks

3.1 Introduction

Originally, IP-based networks provided purely best-effort service; i.e. all data packets were treated equally, with no service level, no bandwidth reservation, nor quality of service guarantee. However, the surge of multimedia, realtime applications brought a need to add QoS mechanisms to the TCP / IP based networks. Consequently, two IP QoS architectures, the Integrated Services (IntServ) and the Differentiated Services (DiffServ), were proposed and deployed. This chapter first reviews the architectural characters of both the IntServ protocol and the DiffServ protocol.

Another very important trend in utilizing today's Internet is the use of a Virtual Private Network (VPN). An IP VPN is defined as an "emulation of a private Wide Area Network (WAN) facility using IP facilities" [12]. The privacy requirements of VPN make security issues more and more important. Most enterprise applications have to be protected by certain cryptographic means, such as encryption and digital signature. Section 3.3 reviews the IETF's IP security standard, the IPSec protocol.

3.2 QoS management in IP networks

Intuitively, any QoS mechanism should have three components. The first component specifies all the supported QoS service levels. Another component is used to classify packets into different service levels. The third component is used to implement

the service levels. The last component in both IntServ and DiffServ is normally involved with buffer management and scheduling, and it can be regarded as the backend of the QoS mechanism. This chapter only covers the first two components in IntServ and DiffServ, which are most closely related to our project. A detailed description of the backend implementation for IntServ and DiffServ can be found in [13].

3.2.1 The IntServ architecture

The IntServ architecture associates each IP packet with a traffic flow. A traffic flow is normally defined by its source address, destination address and port number or protocol type. For every IP flow that needs a non-best-effort service, IntServ requires a reservation to be made before transmission. The most commonly used reservation protocol in IntServ is the Resource Reservation Protocol (RSVP), defined by RFC 2205.

3.2.1.1 QoS services provided by IntServ

Currently, the IntServ architecture defines three categories of services: best effort service, controlled-load service and guaranteed-load service. The default service provided by IntServ is best-effort and it can be used by applications that are not delay-sensitive. Controlled-load service is for delay-sensitive applications that can adaptively tolerate certain amount of delay and jitter. Guaranteed-load service ensures that the end-to-end datagram queuing delay is under a certain limit and is for applications that require bandwidth guarantees and have stringent bounds on delay and jitter [14].

Guaranteed service ensures that datagrams arrive within the guaranteed delivery time and will not be discarded due to queue overflows, provided the flow's traffic stays

within its specified traffic parameters [15]. Controlled-load service “provides the client data flow with a quality of service closely approximating the QoS that the same flow would receive from an unloaded network element, but uses capacity (admission) control to assure that this service is received even when the network element is overloaded” [16].

3. 2.1.2 How packets are served in IntServ nodes

Both guaranteed services and controlled-load services are set up by a reservation protocol, which specifies the traffic characteristics using the TSpec, and the desired service using the RSpec. The TSpec takes the form of a token bucket plus peak rate (p), and a maximum datagram size (M). The token bucket has a bucket depth (b) and a bucket rate (r), and the minimum policed unit (m). All IP datagrams less than size m will be counted as being the size of m when policed and tested for conformance [16]. The XDR description of TSpec (based on [16]) is

```
struct {  
    float r;  
    float b;  
    float p;  
    unsigned m;  
    unsigned M; } TOKEN_BUCKET_TSPEC;
```

The unit for r , p is “byte per second”, while the unit for b , m and M is “byte”.

The RSpec includes a rate R and a slack term S . The rate R is again measured in bytes per second. The slack term signifies the difference between the desired delay and

the delay obtained by using reservation level R . R is a float and S is a 32 bit integer from 0 to $2^{32}-1$ [14].

Edge routers have to ensure that an application using controlled-load service or guaranteed service conforms to its TSpec. According to RFC 2211, a traffic flow “must obey the rule that over all time periods, the amount of data sent does not exceed $rT+b$, where r and b are the token bucket parameters and T is the length of the time period” [15][16]. The unit for r is “byte per second”, and the unit for b is “byte”. This kind of conformance enforcement is called “policing”. In addition to policing, guaranteed service also has “reshaping” operation. Reshaping works by buffering packets until they can be sent in conformance with the TSpec [15]. The buffer size used to reshape traffic is calculated as $b + Csum + (Dsum * r)$. b , r are Token bucket parameters, while $Csum$, $Dsum$ are two error terms representing how the implemented guaranteed service deviates from the ideal fluid model (please refer to [15] for more details). If a packet arrives and the reshaping buffer is full, then the packet is non-conforming and is going to be treated by best-effort service [15].

The above description indicates that the TSpec and the RSpec are used to set up services and are among the configuration variables that constitute context information for an IP micro flow. They are also subject to context information transfer.

3.2.2. The DiffServ architecture

IntServ and RSVP are relatively complex to deploy, and they do not scale well when the traffic amount is large, since they are “per IP flow”-based. Alternatively, the DiffServ architecture has been developed to provide a more scalable QoS solution.

According to the DiffServ view, the networks are composed of multiple DiffServ domains (DS domains). One DS domain is a contiguous set of nodes, all of which have differentiated service implemented [17]. The basic operation of a DS domain includes the following steps:

1. A Service Level Specification (SLS) is established between the DS domain and the customer before providing service
2. The boundary nodes of the DS domain label the IP packets coming from the customer subnet by providing DS codes. The DS code is the leftmost six bits of the Type of Service (TOS) field in the IPv4 header or Traffic Class field in the IPv6 header. A DiffServ boundary node connects one DS domain to another domain. The “labeling” operation is based on pre-configured SLS
3. DiffServ provides an aggregation mechanism to treat all packets with the same DS code equally. The forwarding behavior provided by a DiffServ node to an IP packet with a certain DS code is known as “Per-Hop Behavior” (PHB). Two packets have the same PHB, as long as their DS codes are the same. All nodes in the same DS domain implement consistent PHBs

The structure of a DS domain is shown in Figure 3.1.

Figure 3.1: The structure for a DS domain

3.2.2.1 QoS services provided by DiffServ

In a DS domain, a Per-Hop Behavior (PHB) represents one QoS service level. Three classes of PHB are defined currently. The default is best-effort, which serves delay-insensitive applications. The other two are Assured Forwarding PHBs (AF PHBs) and Expedited Forwarding PHB (EF PHB).

AF PHB groups are specified in RFC 2597 [18] and act as a means of providing different levels of forwarding assurances for IP packets in a DS domain. Four AF classes are defined, and each AF class has its forwarding resource, e.g. bandwidth and buffer size, allocated by a DS domain. Packets in different AF classes are forwarded

independently. Within each AF class, an IP packet is assigned a value representing its drop precedence, and an IP packet with lower drop precedence will be forwarded with higher probability. As mentioned previously, a PHB is represented by a DS code, which is the value contained in the leftmost six bits of the Type of Service (TOS) field in the IPv4 header, or the Traffic Class field in the IPv6 header. The recommended DS codes for the four class, three drop-level AF PHB groups are given in RFC 2597, though DS domains are free to use their own DS codes to select AF PHBs.

EF PHB is used to build a “low loss, low latency, low jitter, assured bandwidth, end-to-end service through DS domains”. DiffServ uses EF PHB to provide “Premium” service, which is comparable to the Guaranteed service in IntServ. RFC 2598 [19] defines EF PHB as “a forwarding treatment for a particular aggregate where the departure rate of the aggregate’s packets from any DiffServ node must equal or exceed a configurable rate”. In other word, EF PHB provides such QoS service that packets are delivered with no queuing delay, as long as they come in agreement with the pre-configured Service Level Specification (SLS). SLS is discussed in 3.2.2.2.

3.2.2.2 How packets are served in DiffServ nodes

The functional elements of DiffServ are made of “a small set of per-hop forwarding behaviors, packet classification functions, and traffic conditioning functions including metering, marking, shaping, and policing” [17]. The functional components of DiffServ are shown in Figure 3. 2.

A packet that enters a DiffServ domain is classified and conditioned at the boundaries of the network, and assigned a DS code. Within the network, a packet is forwarded according to the PHB that is selected by its DS code.

Figure 3.2: Components for a DS boundary node

Traffic Classifiers identify packets from a traffic stream, based on some portions of the packet header. Reference [17] describes two kinds of classifiers. The Behavior Aggregate (BA) Classifier works by looking at the DS code and the Multi-Field (MF) Classifier operates by checking several fields in a IP header, such as source address, destination address, DS field, source port, destination port, and so on. Traffic Classifiers classify packets using some specified rules and associate each packet with a traffic profile, then send them to Traffic Conditioner for further processing. A Traffic profile “specifies the temporal properties of a traffic stream selected by a classifier” and “provides rules for determining whether a particular packet is in-profile or out-of-profile”

[17]. Inside the Traffic Conditioner, packets are first measured by the Meter against their traffic profiles to decide whether they are in-profile or out-of-profile, and then “steered” to different Markers to be given a suitable DS code. The function of the Shaper is to buffer packets temporarily and bring them into compliance with the associated traffic profile. “Droppers discard some or all of the packets in a traffic stream in order to bring the stream into compliance with the associated traffic profile” [17].

Fundamental to the above operations is the Service Level Specification (SLS). SLS is a contract between the service provider and its customers and describes the end-to-end traffic behavior characteristics [20]. SLS has to be configured in DS nodes beforehand since there is no signaling phase in DiffServ. The rest of this section is devoted to a discussion on SLS. The content of SLS is listed as follows.

Scope The scope of an SLS contract indicates where the QoS policy is to be enforced.

Scope = (ingress, egress)

“ingress” and “egress” are interface identifiers or a set of interface identifiers. Scope should uniquely determine the boundary link on which packets arrive / depart at the border of a DS domain.

Flow description Flow description indicates to which IP packets the QoS guarantees are to be enforced.

Flow description = (differentiated service information, source information, destination information, application information).

Differentiated service information is composed of one DS code value or a set of DS code values. Both source information and destination information contains IP address

or a set of IP addresses. Application information can be protocol numbers, or destination ports.

Traffic envelope and traffic conformance The traffic envelope is a set of traffic conformance parameters that describe how the packet stream should look in order to obtain the guaranteed services. As the basic input for traffic conformance algorithms, traffic conformance parameters include:

Peak rate p

Token bucket rate r

Bucket depth b

Maximum transport unit (MTU) M

Minimum packet size m

A traffic conformance test is done at a DS-boundary node (edge router). The conformance test can be binary-based or multi-level based. Binary traffic conformance testing identifies whether a packet is “in-profile” or “out-of-profile”. In a multi-level traffic conformance test, a packet will be tagged as belonging to a particular level (1, 2, ...n). Packets tagged as level n are called “excess”. The value of “n” must be included by SLS.

Excess treatment This relates to how the service provider will process excess traffic.

Performance guarantees It describes the service guarantees that the network offers to the customer for the packet stream described by the flow description. There are four performance parameters: delay, jitter, packet loss and throughput.

Service schedule The service schedule indicates the start time and end time of the service.

Reliability Reliability indicates the maximum allowed average downtime per-year and the maximum allowed time for repair.

SLS is the configuration data that a DS boundary node has to keep in order to work properly. The operations of the Traffic Classifier and Marker in a DS boundary node are based on SLS; consequently, SLS constitutes part of the context information required when processing an IP flow in DiffServ-enabled networks.

3.3 The IPSec protocol

Security has been a very important issue as business-critical applications such as e-commerce become more and more popular. The IPSec protocol [21] is the IETF standard for securing “real-time” applications on the Internet. IPSec operates between the TCP layer and the IP layer, and can be implemented in a host or in a security gateway. The protection provided by IPSec includes access control, authentication, anti-replays and confidentiality. These security services are implemented through the use of Authentication Header (AH) and Encapsulating Security Payload (ESP) [21]. AH contains a cryptographic checksum so that the receiver can check whether the content of the packet has been changed. ESP is used to encrypt the data content of an IP packet. A detailed discussion of AH and ESP is available from [22][23].

Figure 3.3 depicts how protection is provided by the ESP method. In Figure 3.3, the IPSec protocol is implemented in two security gateways: SG1 and SG2. SG1 encrypts all IP packets with IP header1, and then adds the new header (IP header2) to the

encrypted packets. Notice the encryption covers both the original IP header and the data payload. Next the packets enter public networks and are routed according to information stored in the new header, while the data payload and the original IP header are hidden away. These packets are decrypted and their original IP headers are restored in SG2. Figure 3.3 depicts the ESP operation only, but the basic steps are similar when AH protection is used.

The IPsec operations in SG1 are closely related to those in SG2, and, to put it more formally, for each protected traffic flow between SG1 and SG2, there is a security association (SA) between them. According to [21], a SA is “a simplex ‘connection’ that affords security services to the traffic carried by it”. A Security Association (SA) can be viewed as the basic unit of applying protection in IPsec. Inside a security gateway, a SA can be identified by a security parameter index (SPI), a destination IP address and the IPsec protocol name (ESP or AH). SPI is a 32-bit value used for distinguishing packets with the same destination address and the same IPsec protocol. Given an IP packet, how to provide security service is based on the policies stored in the Security Policy Database (SPD).

Figure 3.3: The basic operations of IPsec

The SPD specifies what “services are to be offered to IP datagrams and in what fashion [21]” and contains an ordered list of policy entries, indexed by selectors. A selector can be a destination IP address, a source IP address, a user ID, a transport layer protocol, etc. There are three processing options: discarding the packet, bypassing it with IPSec or affording IPSec to it. If the last option is chosen, the SPD entry must also indicate the applied security service (AH or ESP), related protocols / algorithms, and the associated SAD entry.

In each IPSec implementation there is also a nominal Security Association Database (SAD), the entries of which contain parameters for the afforded security services [21]. There are two types of SAD entries. The first type of entries are for processing outbound packets and they are pointed to by corresponding SPD entries. An outbound packet is one that is leaving the IPSec-enabled node and entering public networks. IPSec services, e.g. encryption, are added to the outbound packets. The other type of SAD entries are for processing inbound packets. An inbound packet comes from public networks and IPSec serves it by removing security services, e.g. decrypting it. The SAD entries for inbound packets are indexed by the outer header’s IP destination address, IPSec protocol (AH or ESP), and SPI. The parameters contained in the SAD are listed as follows.

1. Sequence Number Counter: a 32-bit value used to generate the Sequence Number field in AH or ESP headers.
2. Sequence Counter Overflow: a flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent transmission of additional packets on the SA.

3. Anti-Replay Window: a 32-bit counter and a bit-map (or equivalent) used to determine whether an inbound AH or ESP packet is a replay.
4. Algorithms: keys for AH or ESP processing
5. Security Association Lifetime: a time interval after which the SA must be replaced with a new SA (and new SPI) or terminated, plus an indication of which of these actions should occur.

The above review shows that the operations of the IPSec protocol also depend on the configuration information. The configuration information, such as “what protocol should be applied to a given IP packet,” is stored in the SPD. The operations of IPSec are stateful, i.e. some state variables have to be kept when processing IP packets for one traffic flow. These variables, such as the Sequence Number Counter, Anti-Replay Window and Security Association Lifetime, are stored in the SAD.

3. 4 Summary

This chapter has reviewed three protocols that are required to run real-time, business-critical applications over the Internet. It is clear that in order to make these protocols work properly, some network elements have to keep certain configuration / state variables. For example, both IntServ-enabled nodes and DiffServ domain edge nodes need to know 1) what is the traffic profile for a traffic stream, and 2) what level of service is associated with it. A traffic profile is normally described by a token bucket. IntServ service level can be “best effort”, “control-loaded” and “guaranteed”. DiffServ service level can be “best effort”, “Expedited Forwarding PHB” or one of the “Assured

Forwarding PHBs”. Traffic profile and associated service level, i.e., {**Peak rate, Token Bucket rate, Bucket depth, Service level**}, are examples of protocol configuration information, which is renamed “context information” in the latter part of this thesis.

IntServ, DiffServ and IPSec, however, are designed for wired networks where hosts are static. How to extend them to the mobile world is discussed in the next chapter, together with the formal definitions for context information and context information transfer.

Chapter 4 Context Information Transfer

4. 1 Problem statement: lack of mobility support in QoS, security management protocols

Deploying the IPSec protocol and the DiffServ protocol globally provides a cost-effective solution to building the IP layer Virtual Private Network (VPN), which is a feasible method for many enterprise communications. For example, Figure 4.1 shows how the web videoconference application mentioned in the motivating scenario (Section 1.1) is served using the DiffServ / IPSec-enabled VPN.

As described in Chapter 1, the user in our scenario normally runs the videoconference from his company's local network in Ottawa, using his desktop computer, shown as H1 in Figure 4.1. The company's local network is connected to the public network, where the DiffServ protocol is deployed. For simplicity, we only consider the packets that are sent by H1. The default router for H1 is E0, which supports both the IPSec and the DiffServ. Packets from H1 are first checked against the Security Policy Database (SPD) in E0, which specifies that the Encrypted Security Payload (ESP) protection should be applied. Router E0 encrypts and encapsulates these packets. Next the packets are marked, policed, and shaped according to the DiffServ protocol, using the pre-configured Service Level Specification (SLS) information. Starting from router E1, the packets enter the public network part of the VPN where they are forwarded according to their outer header until they reach the last router in the VPN, i.e. the exit point of the

tunnel (Ez in Figure 4.1). At the exit point of the VPN, the packets are decrypted and may enter another interconnected VPN until they reach the destination.

Figure 4.1: Business-critical applications such as the videoconference application described in our scenario can be served by virtual private networks

VPN in fixed networks is already a proven solution; however, the picture becomes much more complicated when mobility is added. Let's consider what may happen in our motivating scenario when the user moves to Toronto. To simplify the discussion, we will make two assumptions. The first is that wireless users can access the Internet services by Mobile IP, which is supported in both Ottawa and Toronto public networks. The second assumption is that the IPSec and the DiffServ protocols are deployed.

As already mentioned, the user would like to begin the videoconference from Toronto airport using his powerful handheld device, on which Mobile IP protocol is implemented. Figure 4.2 shows that the cellular network in Toronto airport is interconnected to the Internet by edge router Ex. Ex also acts as a foreign agent in the subnet. In mobile IP terminology, a foreign agent is "a router on a mobile node's visited network which provides routing services to the mobile node while registered" [11]. The foreign agent de-tunnels and delivers datagrams that were tunneled by the mobile node's home agent to the mobile node. For datagrams sent by a mobile node, the foreign agent may serve as a default router [11].

With Mobile IP implemented, the user can use his handheld device to access network services such as browsing Yahoo's web site, but he cannot run the business-critical videoconference application. Let's consider the packets sent by the handheld device. First, the required Expedited Forwarding (EF) QoS service will not be provided, since edge router Ex is not properly configured with the Service Level Specification (SLS) for the mobile device. However, a more serious problem is that traffic from the mobile device will be forwarded by Ex without encryption, since Ex is not configured to

provide security service to it. On the other hand, the security gateway at the destination site is expecting encrypted packets and may simply discard unencrypted ones.

Figure 4.2: Sketch for Ottawa and Toronto public networks mentioned in the example scenario

What we need is an infrastructure to pass the network service requirements as well as configuration parameters when a mobile user enters a foreign network and begins a service-centric application. In Figure 4.2 above, for example, if router E0 informs Ex that traffic with certain IP addresses / port numbers is for a videoconferencing application

and should be encrypted and served within a fixed delay limit, then router Ex can update its configuration accordingly to provide the required services. However, even this is not the whole story yet: let's suppose that somehow the configuration information for the videoconference application is passed to router Ex and the application has been successfully started. When the user moves from the airport to the hotel, the videoconference will be disrupted since the new edge router Ey is not configured to provide proper services. In order to minimize the disruptive effect of handover, we also need a mechanism to pass configuration information to Ey. The way Ey gets configured may be different from that by which Ex is configured, since users can tolerate some delay when starting an application but they are much more annoyed if it is interrupted in the middle. The IETF Seamoby working group has proposed context information transfer as a method of passing configuration information (e.g. traffic profile, delay limit, IPSec requirement, etc.) from router Ex to router Ey.

Seamoby's proposal is to be reviewed in section 4.2.

4.2 Context information transfer proposal from the Seamoby working group

4.2.1 Context information definition

According to the IETF Seamoby working group, context information is information "on the current state of a service required to re-establish the service on a new subnet without having to perform the entire protocol exchange" [24]. Most context information comes from the network protocols that are used to provide proper QoS, security or other treatments to IP packets, or to control network access, though protocols

at other network layers, e.g. the Point to Point protocol (PPP) at the data link layer, may also need context information. The protocols currently studied by the Seamoby working group include Authentication, Authorization and Accounting (AAA) protocol, Quality of Service (QoS) protocol, Security protocol, and header compression protocol [24]. Normally, these protocols need configuration data which is acquired from reading pre-installed files, or from running protocols.

There are two categories of context information. The first category of context information is relatively more static, and we call it “configuration context information”. Two examples of configuration context information are given in the follows paragraphs.

DiffServ, as mentioned in Chapter 3, is the scalable IP QoS standard. In a DiffServ-enabled domain (DS domain), differentiated IP forwarding treatments are provided according to the pre-configured Service Level Specification (SLS), the service contract between a DS domain and its customers. A very important part of SLS is composed of traffic profiles. The purpose of one traffic profile is to describe the acceptable traffic behavior for one or a group of IP streams. The content of a traffic profile is normally in token bucket form, with the following parameters: 1) *peak rate*, i.e. the maximal packet coming rate; 2) *token rate*, i.e. the average packet coming rate; and 3) *bucket depth*, i.e. the maximal amount of data that can be buffered in a router. The forwarding service provided by a DS domain can be *Best Effort*, *Expedited Forwarding* and *Assured Forwarding*. The traffic profile (*peak rate*, *token rate* and *bucket depth*) and associated service level is an example of context information.

The IPSec protocol is the IETF standard for securing “real-time applications” on the Internet. As described in Chapter 3, IPSec offers security by using Authentication

Header (AH) and Encapsulating Security Payload (ESP). When an IP packet enters an IPSec-enabled node, the security protection applied depends on the policies stored in the local Security Policy Database (SPD), a nominal database containing an ordered list of policy entries which are indexed by selectors. A selector can be a destination IP address, a source IP address, a user identifier or a transport protocol, etc. A SPD policy entry specifies what security services should be provided to the traffic selected by the selector. There are three possible processing options: discarding the packet, bypassing the IPSec operations, or offering security protections. If the last option is chosen, the SPD policy entry also indicates the applied service (AH or ESP), and related protocols / algorithms. The content of SPD entry, including processing options and encryption algorithm / parameters, is an example of security-related context information.

The second category of context information contains state variables, which are used to re-set up disrupted services in the new router after mobile devices have changed location. We have named this category of context information “runtime context information”. An example of runtime context information is illustrated as follows.

One problem with IP over wireless link for interactive voice conversation is the heavy header overhead. Using Internet protocols, voice data is most likely in the form of RTP packet, which normally has an IP header (20 octets for IPv4), an UDP header (8 octets), and a RTP header (12 octets). In other word, without counting link layer framing, a speech data packet will have 40 octets (60 octets if Ipv6 rather than IPv4 is used), but the size of the payload can be as low as 15 – 20 octets [25]. Researchers have developed several header compression schemes to solve this problem. One solution is ROHC, which

stands for RObust Header Compression [26], and is proposed by the IETF ROHC working group.

It is observed that there is great redundancy between header fields, “both within the same packet header and between consecutive packets belonging to the same packet stream” [26]. Most of the header fields are static throughout the whole process of a session. Take the IPv4 header, for example, among all the fields, *version*, *IHL (internet header length)*, *total length*, *reserved flag (don't fragment flag, more fragments flag)*, *fragment offset*, *protocol*, *source address*, *destination address* are static, or can easily be inferred. These fields shall be compressed away, i.e. the compressor does not have to send these fields any more once the decompressor has received them. Compressor is the sending node over a wireless link. Packets from the compressor contain complete or compressed headers. The Decompressor is the receiving side and restores the compressed header and relays the packets. Conceptually ROHC can be viewed as an “interaction between two state machines”, one being the compressor and the other the decompressor. Both the compressor and decompressor have three states and different header information is exchanged during the different states. ROHC also applies several encoding methods to further compress the length of header fields [26].

Obviously, both compressor and decompressor have to keep the state variables for every IP flow that passes between them. The state information used for header compression and decompression includes 1) static field information, such as source address, destination address and fragment offset and, 2) and dynamic information, such as compressor / decompressor state, compression algorithms and parameters for the algorithms. The ROHC state information is an example of runtime context information.

4.2.2 Local context information transfer

The Seamoby working group proposes that when mobile hosts move from one subnet to another, the interrupted service can quickly be resumed if the related context information, including both configuration information and state variables, is transferred to the new router. The Seamoby working group defines context information transfer as “the movement of context from one router or other network entity to another as a means of re-establishing specific services on a new subnet or collection of subnets” [27]. The IETF Seamoby working group takes context information transfer as a means of decreasing the disruptive effect of handover, and they place emphasis on transferring state variables to re-established interrupted service. According to [24], context information transfer happens mostly in a local area network, where network elements can understand and trust each other. The Seamoby’s context information transfer concept has been renamed “local context information transfer” in this project in order to differentiate it from the concept of “global context information transfer”.

4.3 Global context information transfer

4.3.1 Definition of global context information transfer

Local context information transfer solves the “hand-over caused service disruption” problem. In our motivating scenario, local context information transfer can be applied to pass information from router Ex to router Ey. But we still need a mechanism to configure router Ex before the service-centric application is started. The mechanism proposed by this project is “global context information transfer”.

In order to define “global context information transfer”, we first have to re-interpret the concept of context information. According to the Seamoby group’s definition, context information includes both protocol configuration data and state variables that “snapshot” the operations of network protocols. In this project, the scope of context information is focused on a set of configuration data which captures the operations of one or several network protocols, and these operations are required by a real-time / business-critical application for one or a group of users. We regard context information as a means to describe the service requirements for real-time / business-critical applications.

We define global context information transfer as the action of passing context information from one network element to another remote network element, via the public networks. The sender and the receiver may belong to different management authorities and may not know each other beforehand. The purpose of global context information transfer is to set up network services.

Global context information transfer is proposed as a complement to local context information transfer. The following paragraph illustrates how our motivating scenario could become reality under the global and local context information transfer framework.

When the Ottawa user moved to Toronto airport, he got connected to the local network using his handheld device. After the user selected to start the videoconference application, his handheld device first contacted its home network, asking that the required configuration information, i.e. the context information, be transferred to the foreign network where the user was located, referring again to Figure 4.2. A network element in the mobile user’s home network, e.g. router E0, accepted the request and communicated

with a host located in the Toronto airport network, e.g. router Ex, to send context information. The videoconference application could start after router Ex had updated its configuration using the received context information. Router Ex also kept the context information, and sent it to router Ey after the mobile user entered the hotel. Ey continued to serve the user until the videoconference ended.

We assume that Ex and Ey belong to the same service operator, which implies that mutual understanding and mutual trust are already established. The operations of transferring context information from Ex to Ey fall within the scope of local context information transfer, while the operations of transferring context information from E0 to Ex fall in the scope of global context information transfer.

From the point of view of context information transfer, the Internet can be divided into multiple CIT domains, as shown in Figure 4.3. Every CIT domain is a group of adjacently located network elements that understand the same set of context information transfer messages and use interoperable transferring protocols. We also assume that trust and knowledge, two critical factors for communicating and using context information, are already established among network elements inside one CIT domain via local management means. A CIT domain can overlap with a Mobile IP subnet and stores context information for the mobile nodes whose home network belongs to it. The elements in a CIT domain depend on the network configuration. For example, if mapped to the 3GPP architecture, then a CIT domain can be composed of one or several RAN-GPRS networks, and if mapped to wireless local network (WLAN), a CIT domain may include one or several extended service sets [28] and adjacent wired subnets.

Local context information transfer takes place inside one CIT domain, while global context information transfer happens between two CIT domains, via Context Information Hosts. For one mobile device, the Context Information Hosts located in its home network are its home Context Information Hosts, and the Context Information Hosts located in foreign networks are called foreign Context Information Hosts. Global context information transfer normally happens between a mobile device's home Context Information Host and a foreign Context Information Host.

Figure 4.3: The Internet is divided into multiple CIT domains

4.3.2 Three ways to arrange global context information transfer

There are three possible scenarios which can occur when a mobile device enters a foreign network and plans to begin an application that needs context information transfer.

1. In the first scenario, the mobile device first sends a transfer request message to one of its home Context Information Hosts. The home Context Information Host, in response to the transfer request message, communicates with the respective foreign Context Information Host to send the required context information.
2. In the second scenario, the mobile device first contacts one of the foreign Context Information Hosts. Then the contacted foreign Context Information Host sends a transfer request message to the mobile device's home Context Information Host, asking for context information.
3. In the third scenario, context information is stored in mobile devices. The mobile device sends the context information directly to a foreign Context Information Host before it begins a service-centric application.

We chose to concentrate on the first scenario for this project due to the following reasons:

1. The home Context Information Host and the mobile device are "known" parties, and it is relatively easier for them to authenticate each other.
2. Storing context information in the home Context Information Hosts makes it easy to maintain and update information and decreases the management burdens, which is an important advantage when one user owns multiple devices.

4.4 Global context information transfer and the Virtual Home Environment

The purpose of global context information is to allow mobile users to run service-centric applications with high quality transparently of their locations. In some senses, global context information is comparable to the Virtual Home Environment (VHE) notion [29].

VHE is a very important part of the 3G cellular networks, and it is defined by 3GPP as “a concept for personal service environment portability across network boundaries and between terminals. The concept of VHE is such that users are consistently presented with the same personalized features, user interface customization and services in whatever network and whatever terminal, wherever the user may be located” [29].

Global context information transfer and VHE have one thing in common: both of them deal with service profiles that can be as specific as to one mobile user, and consequently, the profile data plays an important role for achieving global context information transfer or building VHE. However, VHE is an architectural solution while global context information transfer is only a distributed application.

VHE aims for cellular networks, where by tradition, the network operators are more co-operative than their Internet colleagues. Not surprisingly, VHE starts with a set of standards to create an open interface between network operators and service providers. The open interface, called the Application Interface, provides an architectural framework within which services can be created based on standardized service capabilities. The

VHE concept is a critical part of 3G, and if realized, will bring dramatic changes to the ways in which services are provided.

On the other hand, global context information has been proposed as a way to add mobility support to network protocols designed for fixed data networks. Global context information transfer will not cause major changes to the existing network protocols, or the applications that rely on the services provided by these protocols. Global context information can be taken as a network application that communicates service profile data.

Figure 4.4(a) and Figure 4(b) show how context information and global context information transfer can be integrated with the existing Internet architecture.

Current Internet

Figure 4.4(a): Current network protocols and network applications

With context information transfer

Figure 4.4(b): Context Information Transfer can be added at application level

Currently, many network protocols have already been established to serve network-based, service-centric applications. Normally, these service-centric applications are only available to desktop users in fixed networks. Mobility can be added by global context information transfer, which is a distributed application and requires:

1. A communicating mechanism that can pass context information
2. Rendering applications that interpret context information and change-received context information into a format understandable to the local network. A rendering application also interacts with one or several network protocols

4.5 Summary

This chapter has introduced the concept of context information and context information transfer, two phrases that were coined by the IETF Seamoby working group. The original scope of context information transfer has been limited to a local area network, with the goal of minimizing the handover effect.

As a complement to the Seamoby working group's effort, this project proposes using global context information transfer, i.e. communicating configuration context information between two network elements that are connected by public networks. Global context information is one way to pass along critical configuration data when mobile users access services in foreign networks.

Though it sounds straightforward, global context information transfer is faced with two important challenges:

1. The first is Internet heterogeneity. Different network protocols may be deployed in different CIT domains. For example, one CIT domain may have the DiffServ protocol installed, while another may have the IntServ protocol installed. Consequently, both CIT domains can support real-time applications, but they need different configuration data. Even when two CIT domains use the same network protocol, they may still not be able to exchange context information if they have chosen different protocol vendors.
2. Security is the other challenge faced by global context information transfer. Context information, in most cases, is related to network resource allocation. The two Context Information Hosts exchanging context information must be able to authenticate each other, ensure data integrity, and provide privacy if either party requires.

The two challenges are at different levels. The first challenge is a developmental issue, while the second challenge is an operational issue.

The rest of this thesis contains a discussion concerning our solutions to the above two challenges. Chapter 5 discusses describes the creation of an environment in which context information can be published and propagated using the Context Information Publish Framework. Chapter 6 describes in detail the operations of global context information transfer. Obviously, Chapter 5 aims for the first challenge, while Chapter 6 aims for the second challenge.

Chapter 5 Context Information Publishing Framework

5.1 Overview

Although the idea of global context information transfer sounds straightforward, it is a new concept in both the Internet and the mobile computing areas. One major challenge, as we have mentioned in section 4.4, is how to enable context information sent out by one Context Information Host to be correctly interpreted by another Context Information Host belonging to a different management authority. Co-operation among people who define context information provide the key factor to achieving context information understanding; consequently, there is a need for a “behavioral” protocol among people who define context information.

The purpose of the Context Information Publishing Framework (CIPF) is to help create such an open environment so that context information can be communicated / shared among software developers / network operators.

CIPF includes a set of guidelines directing developers to define and publish context information as well as the CIPF application, a Web-based application which facilitates the publishing context information for developers.

Figure 5.1: The three types of entities for defining information

There are three types of entities required when defining information, as shown in Figure 5.1 above.

Concrete data file: As suggested by its name, a concrete data file is a file containing concrete data.

Meta-data file: A meta-data file defines the format for a group of concrete data files. An XML schema document is a meta-data file since it defines the syntax of a group of XML documents. An example of an XML schema document is shown in Figure 5.2. A concrete XML file that obeys the syntax defined in Figure 5.2 is shown as:

```
<?xml version= "1.0" encoding= "UTF-8"?>

<TrafficProfile>

    <MaxRate value= "12.7" type = "kbps" />

</TrafficProfile>
```

Detailed explanations for Figure 5.2 follow in section 5.2.2.

Grammar rules: A set of grammar rules specifying how valid meta-data files can be composed. XML schema technology itself is a set of grammar rules.

Concrete data files that contain context information are used in the operations of transferring context information. CIPF is concerned with how to define the context information files and how to make the definitions known among developers / network administrators.

This chapter is organized as follows. Section 5.2 discusses how to define context information in CIPF. Section 5.3 presents the way in which context information is published in CIPF, and this is followed by the discussion of the CIPF application in Section 5.4.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">

  <!-- defining one element, the element is the top element used in corresponding XML files -->
  <xs:element name="TrafficProfile">
    <xs:annotation>
      <xs:documentation>This is for defining traffic profile</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:choice>
        <xs:element ref="TokenBucket"/>
        <xs:element ref="MaxRate"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <!-- defining the TokenBucket element, which is a subelement of TrafficProfile -->
  <xs:element name="TokenBucket">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="BucketDepth"/>
        <xs:element ref="MaxRate"/>
        <xs:element ref="AverageRate"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- defining MaxRate, which is a subelement of TrafficProfile -->
  <xs:element name="MaxRate" type="ValueUnitType"/>

  <!-- defining BucketDepth and AverageRate, which are subelements of TokenBucket -->
  <xs:element name="BucketDepth" type="ValueUnitType"/>
  <xs:element name="AverageRate" type="ValueUnitType"/>

  <!--type definition. This defines the Value / Unit pair, which is the type of BucketDepth -->
  <xs:complexType name="ValueUnitType">
    <xs:attribute name="Value" type="xs:float"/>
    <xs:attribute name="Unit" type="xs:string"/>
  </xs:complexType>

</xs:schema> <!-- end of schema-->

```

Figure 5.2: The XML schema definition for traffic profile.

5.2 Defining context information in CIPF

CIPF uses XML schema [30] [31] technology to define information. XML schema is the new way of specifying the legal structure for any XML document. The basic functionality of XML schema is similar to that of Document Type Declaration (DTD): they can both specify what XML elements and attributes can appear in the file, where and how many times they can occur, how elements can nest, and how elements can fit together. However, different from DTD, XML schema uses XML syntax, and is more suitable for hierarchical data structures via advanced features such as abstract type and derived types [30]. A specification of XML schema technology is available in [31].

5.2.1 Defining context information

Figure 5.3: Information management in CIPF

Figure 5.3 shows that CIPF information management includes two parts: the information definition part and the information publishing part. A detailed discussion of information publishing follows in 5.3, while this section focuses on information definition.

The artifacts of information definition are XML schema documents defining the configuration data formats for various network protocols (Diff_Serv schema, IP_Sec schema and Int_Serv schema in figure 5.3), and transfer data definition files specifying what information should be included to start a service-centric application (Transfer definition 1, Transfer definition 2 in Figure 5.3).

Two types of meta-data, i.e. *Protocol Configuration Schema* (DiffServ schema, IPSec schema and IntServ schema in Figure 5.3) and *Transfer Data Definition* documents (Transfer definition1 and Transfer definition 2 in Figure 5.3), are handled in CIPF. The *Protocol Configuration Schemas* are used to define the format and the content of configuration data for various network protocols while the *Transfer Data Definition* documents define what configuration information is transferred as a unit for initiating or restoring certain network services.

In CIPF, *Protocol Configuration Schemas* are XML schema documents defined by network protocol developers or network managers. One example of configuration data is the traffic profile for the DiffServ protocol, described in Section 4.2. Figure 5.2 shows the definition of a traffic profile in XML schema terms.

According to Figure 5.2, the content of a traffic profile includes a “TrafficProfile” element, which can contain either a “MaxRate” element or a “TokenBucket” element.

The type of element “MaxRate” is “ValueUnitType”. An element of “ValueUnitType” has two attributes: the “Value” attribute, which is a float and gives the element’s value, and the “Unit” attribute which is a string and gives the unit, e.g. “kilobyte / second”, for the element.

The element “TokenBucket” contains three sub-elements: “BucketDepth”, “MaxRate” and “AverageRate”. As shown in Figure 5.2, the type for all the three sub-elements is “ValueUnitType”.

The XML schema document shown in Figure 5.2, as well as other XML documents presented in this chapter, were created using XML Spy editor, which automatically validates XML files according to the schema or DTD files. Only valid XML documents can be saved using XML Spy editor.

Protocol Configuration Schemas provide the basic building blocks for transferring context information. However, when starting a real-time / business-critical application, information defined in multiple *Protocol Configuration Schemas* may be required. For example, configuration data for both the DiffServ protocol and the IPSec protocol is needed in our motivating scenario. On the other hand, not all elements defined in a *Protocol Configuration Schema* need to be transferred, e.g. “Scope” in SLS (section 3.2.2) is required to configure a DiffServ-enabled domain, but it is not required to restore a service. We need a mechanism to specify what information is to be transferred together for starting a service-centric application in a foreign network. The *Transfer Data Definition* documents serve the purpose.

In CIPF, the *Transfer Data Definition* Documents are a group of XML documents, the syntax of which obey the *Transfer Data Definition Schema*, which is a

XML schema document defined by this project. The *Transfer Data Definition Schema* acts as a standard template for protocol vendors and network operators to specify how context information should be transferred.

Please note that a *Transfer Data Definition* document specifies what information should be transferred as a unit in a context information transfer operation, and, consequently, the *Transfer Data Definition* Document is at the “Meta-data file” level, as shown in Figure 5.4. The *Transfer Data Definition Schema* is at the “Grammar rule” level, since it specifies rules on how to make *Transfer Data Definition* Documents.

Figure 5.4: Relationship between *Transfer Data Definition Schema*, *Transfer Data Definition* Document and content of context information messages

According to the *Transfer Data Definition Schema*, every *Transfer Data Definition* document should contain one top element, the “Context Information Transfer Schema”, the structure of which is shown in Figure 5.5(a).

The element “Context_Information_Transfer_Schema” has three attributes, “ApplicationName”, “AccessURL”, and “DeveloperURL”. The value of attribute

“ApplicationName” identifies the service-centric application for which the *Transfer Data Definition* document is defined. “AccessURL” points to the web page where the *Transfer Data Definition* document can be accessed, while the attribute “DeveloperURL” points to the home page of the developer.

As shown in Figure 5.5(a), a “Context_Information_Transfer_Schema” element contains the sub-elements “Vocabulary” and “TransferUnit”.

One “Vocabulary” element in a *Transfer Data Definition* document identifies one *Protocol Configuration Schema* that is to be referred in the following “TransferUnits” elements. Two required attributes of a “Vocabulary” element are “SchemaFileURL” and “Alias”. “SchemaFileURL” is a URL where the referred *Protocol Configuration Schema* can be accessed. “Alias” is a shortened name for identifying the referred schema. The remaining attributes for the “Vocabulary” element are optional. The “ProtocolDescription” attribute provides a brief description of the network protocol for which the *Protocol Configuration Schema* is defined, and the “ProtocolURL” is the URL where the network protocol specification can be accessed.

One or multiple “Vocabulary” elements are contained in each *Transfer Data Definition* document and are followed by a set of “TransferUnit” elements. Every “TransferUnit” element first specifies the content of context information by including one or several “SchemaElement”s, the type of which is “ElementReferenceType”. “ElementReferenceType” is used to refer a specific XML element that is defined in a *Protocol Configuration Schema*. An element of the type “ElementReferenceType” must have the following two attributes:

1. The “SchemaFileAlias” attribute: it must be equal to the “Alias” attribute in one of the “Vocabulary” elements
2. The “ElementQName” attribute: it must be equal to the qualified name of one XML element that is defined in the schema file referred by the “SchemaFileAlias” attribute

Figure 5.5(a): The structure of a *Transfer Data Document*

By using the “SchemaFileAlias” and “ElementQName” attributes, a “SchemaElement” element refers to an XML element defined in a *Protocol Configuration Schema*.

A “TransferUnit” element may optionally contain “UserInfo” and “ServiceDescription”. “UserInfo” contains information on the users for whom context information is applied. The type of “UserInfo” is also “ElementReferenceType”. “ServiceDescription” provides textual description for the network service.

Figure 5.5(a) is the UML diagram showing the structure of *Transfer Data Definition Documents*, and the XML schema document that defines the structure, i.e. the *Transfer Data Definition Schema*, is listed in Figure 5.5(b).

Based on the grammar listed in Figure 5.5(b), an example of a *Transfer Data Definition* document is shown in Figure 5.6.

Using the “TransferUnit” element, XML document in Figure 5.6 states that two XML elements, the “DiffServ_EF_Level” element and the “MaxRate” element, should be transferred together in order to establish or restore the desired network service. The “Vocabulary” element in the above example indicates that the “DiffServ_EF_Level” element is defined in “http://www.sce.carleton.ca/net_man_lab/DiffServ” namespace and the “MaxRate” element is defined in the following namespace: “http://www.sce.carleton.ca/net_man_lab/TrafficProfile”. Refer to Figure 5.2 and Figure 5.8 for the definition of the “MaxRate” and “DiffServ_EF_Level” elements.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.sce.carleton.ca/net_man_lab/CITSchema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:target="http://www.sce.carleton.ca/net_man_lab/CITSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <!-- Context_Information_Transfer_Schema: top element, the type is defined in
  CIT_Schema_Type -->
  <xs:element name="Context_Information_Transfer_Schema"
    type="target:CIT_Schema_Type"/>

  <!-- defining type for Context_Information_Transfer_Schema element -->
  <xs:complexType name="CIT_Schema_Type">
    <xs:sequence> <!-- sequence of two sub-elements -->
      <xs:element name="Vocabulary" type="target:VocabularyType"
        maxOccurs="unbounded"/>
      <xs:element name="TransferUnit" type="target:TransferUnitType"
        maxOccurs="unbounded"/>
    </xs:sequence>
    <!-- attributes of the element -->
    <xs:attribute name="ApplicationName" type="xs:anyURI" use="required"/>
    <xs:attribute name="AccessURL" type="xs:anyURI" use="optional"/>
    <xs:attribute name="DeveloperURL" type="xs:anyURI" use="optional"/>
  </xs:complexType>

  <!-- type for Vocabulary element: sub-element of CIT_Schema -->
  <xs:complexType name="VocabularyType">
    <xs:attribute name="SchemaFileURL" type="xs:anyURI" use="required"/>
    <xs:attribute name="Alias" type="xs:string" use="required"/>
    <xs:attribute name="ProtocolDescription" type="xs:string" use="optional"/>
    <xs:attribute name="ProtocolURL" type="xs:string" use="optional"/>
  </xs:complexType>

  <!-- type for TransferUnit element: sub-element of CIT_Schema element -->
  <xs:complexType name="TransferUnitType">
    <xs:sequence>
      <xs:element name="UserInfo" type="target:ElementReferenceType" minOccurs="0"/>
      <xs:element name="ServiceDescription" type="xs:string" minOccurs="0"/>
      <xs:element name="SchemaElement" type="target:ElementReferenceType"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <!-- type for UserInfo, SchemaElement -->
  <xs:complexType name="ElementReferenceType">
    <xs:attribute name="SchemaFileAlias" type="xs:string"/>
    <xs:attribute name="ElementQName" type="xs:string"/>
  </xs:complexType>

</xs:schema>

```

Figure 5.5(b) Transfer Data Definition Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- top element -->
<Context_Information_Transfer_Schema
xmlns="http://www.sce.carleton.ca/net_man_lab/CITSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
ApplicationName= "realTimePlayer"
AccessURL= "http://www.sce.carleton.ca/net_man_lab/example1" >

    <Vocabulary
        SchemaFileURI="http://www.sce.carleton.ca/net_man_lab/DiffServ"
        Alias="ServiceLevel"/>

    <Vocabulary
        SchemaFileURI="http://www.sce.carleton.ca/net_man_lab/TrafficProfile"
        Alias="TrafficProfile" />

    <TransferUnit>
        <SchemaElement SchemaFileAlias="ServiceLevel"
            ElementQName="DiffServ_EF_Level"/>
        <SchemaElement SchemaFileAlias="TrafficProfile"
            ElementQName="MaxRate" />
    </TransferUnit>

</Context_Information_Transfer_Schema>

```

Figure 5.6: An example of a *Transfer Data Definition* Document

According to the *Transfer Data Definition* document defined in Figure 5.6, the following lines can be included in a message containing context information:

```

<DiffServ_EF_Level>gold </DiffServ_EF_Level>

<MaxRate Value = "122.7" Unit = "kilo byte / second"> </MaxRate>

```

5.3 Publishing context information

5.3.1 Review of UDDI technology

As shown in Figure 5.3, CIPF information management includes information definition and information publishing. The motivation for information publishing is to let developers share context information definition files globally.

XML schema technology only provides definitions, and in order to be globally accessible, these definition files are published using Universal Description, Discovery and Integration (UDDI) technology [32][33].

UDDI was jointly developed by more than 300 companies, including major platform and software providers, marketplace operators and e-commerce leaders. The goal of UDDI is to develop an industry-wide infrastructure for organizations to register their services and to quickly discover other services using Simple Object Access Protocol (SOAP)-based technologies. In the UDDI realm, information on enterprise web services is stored in a Business Registry, which by itself is also a Web service which accepts information-publish and information-query messages using SOAP [35].

UDDI specifications cover the following three aspects:

1. Core information types: The four core information types, including *business information*, *service information*, *binding information* and *information on service specifications*, are defined by XML schema in [32]. Any UDDI-compliant Business Registry must allow businesses to register the above four types of information.
2. API for accessing Business Registry: [33] specifies both Publisher API and Inquiry API that must be exposed by all instances of UDDI Business Registry.
3. Business Registry management: The Custody Transfer protocol is required to be implemented by a UDDI Business Registry in order to support moving registry entries from one Business Registry instance to another. The Custody Transfer protocol, as well as other behavioral requirements for Business Registry, is specified in [32].

In addition to services, meta-data can also be published in the UDDI registry as tModel entities. tModel is the abbreviation for Technical Model, and it is the basic mechanism in UDDI to represent “unique concepts or constructs,” including specifications, protocols or namespaces. A given “tModel” entity is uniquely identified by its “tModelKey”, a URI provided by the “tModel” element publisher or the UDDI registry.

A tModel entity can be viewed as a “meta-data” element for a “meta-data” file. The following elements are contained in a tModel entity [34]:

1. A mandatory “uddi:name” element that provides an understandable name of the tModel entity
2. An optional group of “uddi: description” elements that provide textual descriptions for the tModel entity
3. An optional group of “uddi:overviewDoc” elements that contain textual description and / or a URL to provide overview information about the defined concept
4. An optional “uddi:identifierBag” element that may contain references to other “tModel” elements
5. An optional “uddi:categoryBag” element that has a similar structure to the “uddi:IdentifierBag” but with the purpose of providing categorization information

The category information for a tModel entity is contained in the “categoryBag” element. UDDI provides flexible taxonomy mechanisms to categorize the published

entities. Among the eight categorization schemes supported by UDDI 3.0 [34], the **UDDI General Keyword Category System** and **UDDI Derived From Category System** are used by CIFP.

The **General Keyword Category System** provides a simple, lightweight means of establishing and using unchecked UDDI category systems. Using the **General Keyword Category system**, a tModel entity can be categorized using value sets defined in any namespace. Such value sets are generally simple and often of interest only to a small number of people. For example, the following “CategoryBag” element is used to categorize a tModel entity by “DiffServ”, a keyword that is defined in the namespace “http://www.sce.carleton.ca/net_man_lab/PCatSchema”.

```
<categoryBag>
  <keyedReference
    keyName="http://www.sce.carleton.ca/net_man_lab/PCatSchema"
    keyValue="DiffServ"
    tModelKey="uddi:uddi.org:categorization:general_keywords"/>
</categoryBag>
```

The above “keyName” attribute specifies the namespace where the keyword is defined. The “keyValue” attribute is the value of the keyword categorizing the currently defined concept. The “tModelKey” attribute refers to the **General Keyword Categorization** tModel entity, a canonical tModel entity defined by the UDDI specification. Please note that “uddi:uddi.org:categorization:general_keywords” is the *tModelKey* of UDDI version 3; the old version value is “uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4”.

Figure 5.7: tModel entity X is derived from tModel entity Y

The **UDDI Derived From Category System** is used to associate one tModel entity with some other tModel entities for the purpose of extension and redefinition.

Figure 5.7 shows an example of how one tModel entity can be derived from another. tModel entity X is published for a QoS configuration schema. A DiffServ configuration schema is defined by reusing and extending some definitions in the QoS configuration schema, and a tModel entity Y is created for the DiffServ configuration schema. The relationship between tModel entity Y and tModel entity X can be described using the following categoryBag element:

```

<categoryBag>
  <keyedReference
    keyName="Derived QoS"
    keyValue="uddi:ubr.uddi.org:contextInformation:protocol:QoS"
    tModelKey="uddi:uddi.org:categorization:derivedFrom"/>
</categoryBag>
  
```

The above “tModelKey” attribute refers to the **UDDI Derived From Category** tModel entity, which is a standard tModel entity supported by all the UDDI registries.

The “keyValue” attribute refers to the tModel entity from which the current one is derived, i.e. the QoS toModel entity in our case.

CIPF recommends publishing context information meta-data as tModel entities in UDDI public registries.

The purpose of information publishing is to propagate the meta-data files. tModel entities, rather than the definition documents, are published in a public registry. A tModel entity provides enough information to describe and categorize a concept or a namespace.

The purpose of publishing tModel entities is twofold:

1. Public registry URL plus tModelKey value provides a unique identifier for identifying a meta-data file. One way of negotiating what context information should be transferred is by communicating the URL and tModelKey (section 6.4.2). However, please note that CIPF does not recommend that Context Information Hosts get tModel entities from UDDI registries at runtime since accessing a public registry is very slow.
2. The readers of tModel entities are protocol developers and network administrators. It is hoped that by browsing tModel entities, protocol developers and network administrators can discover “what context information has been defined for what application”.

Section 5.3.2 discusses how to publish tModel entities for context information.

5.3.2 Publishing tModel entities in UDDI registry

For every *Protocol Configuration Schema* and *Transfer Data Definition* document, a tModel entity is published in the UDDI public registry. In addition to basic information such as tModel publishing name and overview URL, tModel developers also categorize the tModel entities using the **UDDI General Keyword Category System** or **UDDI Derived From Category System**.

We have defined a set of keywords to facilitate categorizing the *Protocol Configuration Schemas*. These keywords include “QoS”, “Differentiated Service”, “Integrated Service”, “Security”, “IPSec”, “IPSec_ESP”, “IPSec_AH”, “Authentication”, “Authorization”, “Account”, etc., and are associated with the namespace “http://www.sce.carleton.ca/net_man_lab/PCatSchema”. A tModel entity has been composed for the above namespace and is published in the IBM UDDI Business Test Registry, located at “<https://uddi.ibm.com/testregistry/registry.html>”, with the tModel key as “UUID:613978B0-289A-11D7-9B59-000629DC0A53”.

Figure 5.8 shows an XML schema definition for the Expedited Forwarding service in the DiffServ architecture, which is a typical example of the *Protocol Configuration Schemas*. The corresponding tModel entity is shown in Figure 5.9.

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="DiffServ_EF_Level">

    <xs:annotation>
      <xs:documentation> Defining service level of diffserv </xs:documentation>
    </xs:annotation>

    <xs:simpleType name="EF_Level_Enum">
      <xs:restriction base="xs:string">
        <xs:enumeration value="Gold"/>
        <xs:enumeration value="Silver"/>
        <xs:enumeration value="Bronze"/>
        <xs:enumeration value="Default"/>
      </xs:restriction>
    </xs:simpleType>

  </xs:element>

</xs:schema>

```

Figure 5.8: The XML schema definition for the DiffServe EF service level

```

<?xml version="1.0" encoding="UTF-8"?>
<tModel xmlns="urn:uddi-org:api_v3" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <name>DiffServLevel</name>

  <overviewDoc>
    <description>This is the definition for DiffServ service level</description>
    <overviewURL>http://www.sce.carleton.ca/net_man_lab/DiffServ </overviewURL>
  </overviewDoc>

  <categoryBag>
    <keyedReference tModelKey="uddi:uddi.org:categorization:general_keywords"
      keyName="http://www.sce.carleton.ca/net_man_lab/PCatSchema" keyValue="DiffServ">
    </keyedReference>
  </categoryBag>

</tModel>

```

Figure 5.9: The tModel element for the DiffServ EF service level definition

The tModel entities for *Transfer Data Definition* documents are composed in a similar fashion; however, in addition to the “categoryBag” element, tModel entities for

Transfer Data Definition documents also contain “identifierBag” elements, the purpose of which is to indicate what other namespaces have been re-used. The structure of the “identifierBag” element is similar to the “categoryBag”. Using the “identifierBag” elements, we can show what *Protocol Configuration Schemas* are referred by the *Transfer Data Definition* documents.

5.4 The CIPF application

5.4.1 The purpose of the CIPF application

The previous section describes how developers can define and publish *Protocol Configuration Schemas / Transfer Data Definition* documents on the Internet. Context information is business-critical data and it must be authenticated before being published in public networks. UDDI tModel technology provides a digital signature element to assure the data authenticity and integrity [34]. However, there is a scalability problem with this arrangement: on a global scope, it is not realistic to assume two arbitrary parties own enough information to verify each other’s digital signature. One possible solution is to use a broker agent, which signs the data on behalf of the publishers. Data users need only verify the broker agent’s digital signature. The digital signature normally has to be updated when the public key is changed. Another advantage to using a broker agent is that the developers do not have to maintain the tModel entity digital signatures.

We have designed a CIPF application which can be installed in the broker agents. The CIPF application accepts information on *Protocol Configuration Schemas* and *Transfer Data Definition* Documents as well as authenticates developers. Another functionality of the CIPF application is to check reference validity. The most important

principle for defining context information is data sharing, and CIPF encourages one definition file to refer the elements defined in other files. The CIPF application can check whether the referred files are accessible from the Internet.

The functionality of the CIPF application is described by use cases.

5.4.2 Use case description for the CIPF application

The major use case for the CIPF application, the tModel publish use case, is used to describe how the application accepts input from users and publishes tModel entities on behalf of users. The textual description of this use case follows the use case description template given in [36]. The content of the template includes:

1. Use case name: The formal name of the use case
2. Summary: A very brief summary of the purpose for the use case
3. Actor: The name of the Actor who interacts with the system
4. Preconditions: The conditions before the use case begins
5. Descriptions: Scenario description of the use case
6. Alternatives: Alternative paths of the scenario
7. Post conditions: The effect left in the system after the use case
8. Comments: Anything that is regarded of significance for the use case

Description of use case

Use Case Name: tModel Publish Use Case

Summary: Developers publish tModel entities for a *Configuration Data Schema* or a *Transfer Data Definition Document*

Actor: Context information developers

Precondition: The actor has access to the Internet and has a Web browser

Description:

1. The system prompts the actor to choose publishing a tModel entity or composing a tModel entity online
2. The actor chooses composing a tModel entity online
3. The system prompts the actor to choose composing a tModel entity for a *Configuration Data Schema* or for a *Transfer Data Definition Document*
4. The actor chooses to compose a tModel entity for *Configuration Data Schema*
5. The system prompts the actor to enter information such as the name of the published tModel entity, the access URL for the definition file, and the categorization information
6. The actors input information and click the submit button
7. The system composes the tModel entity based on the actor's input and shows the tModel entity to the actor
8. The system prompts the actor to input the private key and public key certificate for signing the tModel entity
9. The actor inputs the private key and the public key certificate and pushes the submit button
10. The system checks the validity and integrity of the tModel entity and shows the result of this check to the actor

Alternatives:

1. If the actor has already defined the tModel entity by himself, he can choose to publish tModel entity in step 2. In this case, the system prompts the user to input the tModel entity in XML format.
2. The actor can choose to publish the tModel entity for *Transfer Data Definition* documents in step 4.
3. If the actor chooses to publish the tModel entity for a *Transfer Data Definition* document in step 4, the system prompts the actor to enter the referred XML schema files in addition to the information listed in step 5.

Postcondition: The tModel entity is published in both UDDI registries and the internal database with an associated digital signature provided by the system.

In addition to the tModel Publish Use Case, the fully developed CIPF application also has other use cases, including:

1. tModel Delete Use Case for removing tModel entities from UDDI registry and internal database
2. tModel Digital Signature Update Use Case for updating the digital signature for all or a subset of the tModel entities
3. tModel Search Use Case for searching tModel entities published through the CIPF application, and for searching tModel entities by category keywords.

These use cases are not covered in detail since they are not part of the prototype implementation. However, as is shown in the next section, the flexible design makes it easy to extend the application to support these use cases.

5.4.3 Design and prototype of the CIPF application

5.4.3.1 Overview

The CIPF application that supports the tModel publish use case should provide the following functions:

- F1: Compose tModel entities for both *Protocol Configuration Schemas* and *Transfer Data Definition* documents.
- F2: Authenticate users based on Public Key Infrastructure
- F3: Make Digital signatures for tModel entities
- F4: Add the tModel entities to the UDDI public registries
- F5: Store the tModel entities in database

Only F1, F2, F3, and F5 were implemented in this project. F4, i.e. adding the tModel entity to public registries, is one of the most important functions expected from the CIPF application; however, at the time the thesis was written, most of the public UDDI registries are only compatible with UDDI v2.0 and don't support digital signatures for tModel entities. A digital signature inside a tModel entity is a new feature of UDDI V3.0. Since the goal of the CIPF application is to compose a digital signature on behalf of developers, it has been decided that F4 should be left for future work, when UDDI v3.x registries become popular.

Please note that delaying the implementation of F4 did not affect the research in this project, especially the global context information transfer operation discussed in the next section, since the published tModel entities are not necessarily required in global context information transfer. As has been emphasized, the most important purpose of using UDDI is to allow developers to share information.

The CIPF application has been designed as a Web-based, three-tier application, as shown in Figure 5.10.

Figure 5.10: Design for the CIPF application

The front end of the CIPF application is the Web-tier, which interacts with clients by HTTP and generates HTML documents. The functions of the Web-tier include composing tModel elements, verifying data integrity, authenticating users, and making digital signatures. The business-tier of the CIPF application provides value-added functions such as reference checking and digital signature updating. tModel elements are stored by the data-tier, in both public UDDI registries and databases belonging to the CIPF application.

As described previously, one purpose of the CIPF application is to make digital signatures for the tModel entities published by authenticated users; consequently, our prototype is concerned with security-related issues, including authenticating users and making digital signatures. The XML digital signature and XML key management technologies are used in our implementation.

5.4.3.2 XML digital signature technology and XML key management technology

The data-security related operations in the CIPF application are based on public key cryptography, which is briefly reviewed in section 6.1. The prototype implementation makes use of two XML security technologies.

The XML signature standard provides a uniform way of communicating digital signature values, as well as the information required to verify the signature. According to XML digital signature specification [37], the root element of a XML digital signature message is the “Signature” element, which contains the following four sub-elements:

1. “SignatureValue” element: This element contains the digital signature value

2. “SignatureInfo” element: This element contains information on how the digital signature is made by including the hashing and the signing algorithms in the “DigestMethod” and the “SignatureMethod” sub-elements.
3. “KeyInfo” element: The “KeyInfo” element is optional, and contains the public key that is used to verify the digital signature. The Elements “DSAKeyValue” and “RSAKeyValue” are defined by [37] to communicate DSA and RSA public keys. The “KeyInfo” element also contains information about the public key certificate. The purpose of the public key certificate is to bind a public key with the identity of the key owner. Further review on public key cryptography and certificates are given in section 6.1.
4. “Object” element: An “Object” element is an optional element that may occur one or several times. An “object” element can contain any data.

Verifying a digital signature includes the following two steps:

- Step 1: Retrieving the public key and verifying the binding of the public key with the subject which owns the public key
- Step 2: Verifying the digital signature using the public key.

Step 2 involves purely mathematical calculation; however, the operations of step 1 need the underlying Public Key Infrastructure (PKI). A PKI [38] can be loosely defined as the set of infrastructure services that support the wide-scale use of public key cryptography. The term PKI normally covers certification authorities (CA). A CA issues and manages certificates for a population of public-private key-pair holders. Each

certificate contains a public-key value and information that unambiguously identifies the certificate's subject, i.e. the person, device or entity which holds the corresponding private key. Every certificate is digitally signed with the CA's signing private key. A certificate reliably binds a public key value with the identity of the key-holder and it can be distributed publicly.

The most widely used standard public-key certificate format is the X.509 certificate, which is defined in the ITU X.509 standard. The standardization of X.509 PKI is currently being undertaken by the IETF [39]. In addition to the X.509 certificate, a few alternative formats have been specified, such as the PGP certificate and simple PKI (SPKI).

Programming with PKI is tedious. To simplify the integration of PKI and XML security technologies and free programmers from low-level programming with PKI, W3C has sponsored the XML key management technology. The mission of this XML management technology is to develop a "specification of XML application/protocol that allows a simple client to obtain key information (values, certificates, management or trust data) from a Web service"[40]. Using XML digital signature and XML key management technologies, software developers can delegate the interaction with PKI and even the processing of digital signatures to dedicated Web servers.

XML key management specification [40] includes two parts: (1) the XML Key Information Service Specification (X-KISS) and (2) the XML Key Registration Service Specification (X-KRSS). X-KRSS is a protocol for registering public key information, while X-KISS defines a protocol to delegate the processing of the "KeyInfo" element sent with a digital signature to a dedicated web server.

The CIPF application is designed under the XML security framework, as shown in Figure 5.11.

Figure 5.11: The CIPF application delegates part of the digital signature processing to dedicated trust service server

Figure 5.11 shows that data sent from the Client to the CIPF application is protected using public key cryptography, in the form of an XML digital signature. The CIPF application delegates part of the digital signature processing, e.g. verifying the public key contained in the digital “Signature” element, to a dedicated trust service server. A trust service server is a Web server that supports XML key management and provides services such as registering a public key and resolving public key information.

The operations of a trust service server rely on local data as well as the underlying PKI.

In our prototype implementation, we have used VeriSign's XKMS Interoperability Service as the trust service server, the URL for which is "<http://interop-xkms.verisign.com/xkms/Acceptor.nano>".

VeriSign's XKMS Interoperability Service is a trial implementation of the XML key management specification. It supports the X-KRSS operations, including key registration and revocation, as well as the X-KISS operations, including public key validation.

VeriSign has also developed a "Trust Service Integration Kit"(TSIK), a group of APIs for using the XKMS Interoperability Service. TSIK provides implementations a client may need when communicating with a trust service server. Prototype implementation of the CIPF application has used the TSIK APIs downloaded from the following web page: "<http://www.xmltrustcenter.org/xkms/developer/index.htm>".

5.4.3.3 Design and prototype of the presentation tier

The first tier is the presentation tier, which normally includes presentation components and front components. The presentation components generate the HTML / XML responses that constitute the user interface, and front components handle HTTP requests and determine what presentation components should be called. Java Servlet and JavaServer Page (JSP) [41][42][43] have been chosen to implement the presentation tier at the server side. Compared with the traditional Common Gateway Interface (CGI), Servlet and JSP technology are more efficient and scalable. They can be deployed to any platform where Java Runtime is installed. Java Servlet is more suitable for implementing

the front components, while JSP provides a very convenient way of implementing the presentation components.

Figure 5.12 is a UML class diagram showing all the JSP instances, the Java Servlet classes and the HTML pages that were implemented by this project. In our implementation, two classes, Publisher and InputProcess, were stereotyped as the front controller, while the rest were stereotyped as presentation components.

Both Publisher and InputProcess are subclasses of `java.servlet.http.HttpServlet`, the ancestor of all the HTTP Servlet classes.

The class Publisher overrides the `doGet()` and `doPost()` methods of `java.servlet.http.HttpServlet`, and accepts HTTP requests from Selection Page, a static HTML page. The Selection Page prompts the user to select whether to publish the tModel entity for a *Protocol Configuration Schema* or for a *Transfer Data Definition document*, and also allows the user to choose whether to compose the tModel entity on-line or not.

Figure 5.12: The design of the presentation tier in the Context Publish Framework

If the user chooses to compose the tModel entity for a *Protocol Configuration Schema*, then SchemaTMComposer is presented to collect the data. SchemaTMComposer is a JSP instance and generates the GUI, as shown in Figure 5-13.

The screenshot shows a web browser window titled "Input Information for The Published Schema - Microsoft Internet Explorer". The address bar shows "A:/PPub.html". The main content area has the heading "Collect Data To Publish The tModel Entity For *Protocol Configuration Schema*".

The form contains the following fields and controls:

- "The published name for the tModel entity:" followed by a text input field.
- "The URL for the schema:" followed by a text input field.
- "Select the keywords for the schema:" followed by a list box containing "QoS", "QoS DiffServ", "QoS IntServ", and "Security".
- A section titled "If other is selected, please specify:" containing:
 - "The keyword:" followed by a text input field.
 - "The tModelKey for the namespace:" followed by a text input field.
 - Buttons: "More keyword", "Delete all", and "View all".
- "The parent schemas:" followed by a text input field.
- Buttons: "More parent schema", "View all", and "Delete all".
- A "Submit Information" button at the bottom right of the form area.

The Windows taskbar at the bottom shows the Start button, several open applications, and the system clock displaying "5:23 PM".

Figure 5.13: The GUI for publishing tModel entities for Protocol Configuration Schemas

If the user chooses to compose the tModel entity for a *Transfer Data Definition* document, then a TransferTMComposer is presented. The GUI generated by TransferTMComposer is shown in Figure 5-14.

Collect Data To Publish The tModel Entity For Transfer Data Definition

General Information:

Publish name for the tModel entity.

The URL for the Transfer Data Definition.

Brief description for the Transfer Data Definition(optional)

Add References:

Select the reference type

User Information ☐ Context information content ☐ Keyword category ☐ Parent category ☐

The access URL or the tModelKey for the reference Namespace URL ☐ tModelKey ☐

The elements referred More elements

More references View all reference info

Submit transfer unit data

Figure 5.14: The GUI for publishing tModel entities for Transfer Data Definition documents

The Class Publisher accepts the tModel entity-related data from SchemaTMComposer and TransferTMComposer to compose the tModel entities, and then the Publisher passes the HTTP request, together with the composed tModel entity to PlainInput, another JSP instance as shown in Figure 5.12.

PlainInput contains a Java Applet, DSApplet, which is designed to provide a digital signature for the composed tModel entity at the client side. DSApplet prompts users to input the private key and the corresponding public key certificate, and then it composes a XML digital signature. The tModel entity, the attached digital signature and the public key certificate are transferred together as a XML digital signature document to class InputProcess, which verifies the digital signature. InputProcess also makes a new digital signature using the server's public / private key pair for any tModel element that has a valid digital signature.

InputProcess and DSApplet use classes in “com.verisign.xmlsig”, a package in TSIK downloaded from “<http://www.xmltrustcenter.org/xkms/developer/index.htm>”, for signing and verifying digital signatures. The class InputProcess also has to validate the public keys sent by the users. As mentioned previously, validating a public key, i.e. ensuring the correct binding of a public key with its subject, needs to interact with the underlying PKI. Class InputProcess delegates the public key validation task to VeriSign's XKMS Interoperability Service server (“<http://interop-xkms.verisign.com/xkms/Acceptor.nano>”).

Class “com.verisign.xkms.client.XKMSValidate”, a class belonging to TSIK, implements methods for composing public key validating message and communicating

with the trust service server. InputProcess uses the following method of “com.verisign.xkms.client.XKMSValidate”:

```
public XKMSValidate(java.security.cert.X509Certificate cert,  
    java.lang.String[] responses)  
    throws XKMSIllegalArgumentException, XKMSException
```

The in-parameter cert is the client’s certificate, while the in-parameter responses contains an array of string constants to indicate what key information is requested from the server.

The key validating request message can be sent out by calling the following method of class “com.verisign.xkms.client.XKMSValidate”.

```
public XKMSValidateResponse sendRequest(XmlTransport transport)  
    throws java.io.IOException, XKMSMalformedResponseException,  
    XKMSSignatureVerificationException, XKMSServiceFailureException,  
    XKMSIllegalArgumentException, InsecureTransportException,  
    XKMSException, XmlMessageException
```

The type for in-parameter transport is “com.verisign.messaging.XmlTransport”, an interface for all the transport classes in TSIK. InputProcess uses an instance of “com.verisign.messaging.XmlTransportNaked” when sending the key-validating message. A detailed API documentation for the TSIK classes is available from the following web page:

“<http://www.xmltrustcenter.org/xkms/developer/index.htm>”.

Once InputProcess finishes processing a tModel entity, it passes the tModel entity and the associated digital signature to the business tier, which is Enterprise JavaBeans (EJB)-based.

5.4.3.4 Design and prototype of the business tier

EJB is the architecture for component-based, distributed computing in the Java platform [44]. Enterprise beans are the components that operate in an environment compatible with EJB specifications. Such an environment is called a EJB container. The EJB container controls all aspects of Enterprise beans' execution, including lifecycle, transaction and persistence. Any Enterprise bean has a set of fixed interfaces that are defined for its client application, i.e. the client-view contracts. The client-view contracts provide a uniform accessing model for applications that use enterprise bean components. The client-view contract includes how an enterprise bean can be created, discovered and removed, as well as all the bean's business methods that can be invoked by client applications. The enterprise bean provider and the container cooperate to create the Enterprise bean's client-view contract in the following manner:

1. The enterprise bean provider specifies the enterprise bean's "home interface" which defines the methods for the client to create, remove and find EJB components. The container creates a class that implements the home interface.
2. The enterprise bean provider also defines the bean's "component interface" to specify all the client-callable business methods. The bean container creates a class that implements the component interface [44].

The client-view contract shields the real implementation of the component, and consequently, EJB systems have two other advantages. First, new applications can be easily incorporated into the system. Second, the deployment of enterprise beans can be changed without disturbing client applications.

In our prototype implementation, only one entity bean, the Storage bean, has been implemented. However, the well-encapsulated nature of EJB makes it easy to add new enterprise bean components without disturbing the existing system. It is expected that enterprise beans with the following functions be added for the fully-developed CIPF application:

1. Reference checking: Checking whether the refereed elements in a *Transfer Data Definition* document exist or not
2. Digital signature updating: Updating the digital signatures for the tModel entities published via the CIPF application
3. tModel entity registering: Publishing tModel entities to public UDDI registries
4. tModel entity searching: Searching tModel entities using categorization keywords.

The Storage bean is an entity bean for storing tModel entities and the associated digital signatures. The Storage bean is designed as a Container Managed Persistent entity bean. EJB support two types of persistence: the Bean-Managed Persistence and the Container-Managed Persistence. In the Bean-Managed Persistence, the enterprise bean component has to implement the database accessing methods, while in the Container-Managed Persistence, the EJB container handles the database access required by an entity bean [44]. We select to implement the Storage bean using Container Management Persistence to make the application portable to several kinds of databases.

As shown in Figure 5.15, the Storage Bean component has the following four interfaces:

1. Remote home interface:

2. Remote component interface:
3. Local home interface:
4. Local component interface:

Since EJB 2.0, the local interface has been supported to increase efficiency. In EJB 1.1, both the home interface and component interface are remote, i.e. the enterprise bean clients access the services provided by enterprise beans only through java remote method invocation (RMI). Starting from EJB 2.0 [45], the enterprise beans have begun to expose local interfaces in addition to remote interfaces. The local interface allows lightweight service access and it is designed for clients that are in the same EJB container.

Figure 5. 15: The Storage bean component

The interaction of the first tier and the second tier in the CIPF applications is via InputProcess and Storage Bean. Since the first tier and the second tier may be deployed in different hosts, InputProcess calls the methods exposed by the remote home interface. The local interfaces of the Storage bean are created for other enterprise beans that are to be added later.

5.4.3.5 Development tools

The Web tier of the CIPF application was developed and tested using Java Web Service Developer Package (JWSDP), an integrated toolkit containing:

1. Apache Tomcat 4.1.2 container
2. Ant Build Tool 1.5.1
3. XML-related APIs

JWSDP was downloaded from the following web page:

“<http://java.sun.com/webservices/download.html>”.

The Storage bean was developed using the Java 2 Software Development Kit, Enterprise Edition (J2EE SDK), a binary version of the J2EE reference implementation, plus related development tools and documentation that enable programmers to develop distributed applications for the J2EE platform. The integration of the first tier and the second tier was also tested using J2EE SDK.

J2EE SDK was downloaded from the following web page:

“<http://java.sun.com/j2ee/download.html>”.

5. 5 Summary

This chapter describes CIPF, a framework dedicated to the method of fulfilling the first requirement of global context information, i.e. how a semantic understanding of context information can be achieved between hosts belonging to different management authorities.

According to CIPF, a mutual understanding of context information comes from 1) defining context information using an open, human-friendly language, and 2) publishing context information definitions in public registries. CIPF relies heavily on XML technologies, though other standards such as ASN.1 can co-exist under CIPF.

The artifacts of CIPF include 1) a set of XML definition documents that define context information and specify what information should be transferred together to serve a service-centric application, and 2) a set of tModel entities that summarize the XML definition documents. It is hoped that by browsing tModel entities, software developers can share and reuse context information definition files, while network administrators can decide what context information definition files should be supported in their local networks.

This chapter has also described the CIPF application, a Web portal designed to facilitate publishing context information. XML digital signature and XML key management, two relatively new XML technologies, have been used in our prototype. XML security technologies make it possible to delegate security-related operations to dedicated Web servers, which makes applications more scalable and maintainable in the long run. However, our unpublished data showed that it took much longer to process XML digital signatures and XML encryptions than it took for traditional digital signature and encryption. That is why they were used for the CIPF application prototype, but not recommended for global context information transfer, which is discussed in the next chapter.

Chapter 6 illustrates one solution for carrying out global context information. A basic assumption used in the discussion in the next chapter is that two remotely located

Context Information Hosts are already properly configured by their network managers, using the facilities provided by CIPF.

Chapter 6 Global Context Information Transfer

6.1 Overview

As described in section 4.3, global context information transfer takes place between different CIT domains. Every CIT domain is served by one or multiple Context Information Hosts, which act as the application gateways for global context information transfer. Section 4.3 also describes three possible scenarios on how context information can be communicated. This chapter covers the implementation of the first scenario. An important assumption for this chapter is that proper Transfer Data Definition Documents (section 5.2) and rendering applications (section 4.4) have been installed in Context Information Hosts using the facilities provided by CIPF.

Figure 6.1 shows that the first global context information transfer scenario is divided into three phases.

Figure 6.1: The three phases in Global Context Information Transfer

The first phase is the service discovery phase. During the service discovery phase, mobile devices discover the foreign Context Information Hosts, i.e. the application gateways for accepting context information in visited networks. The second phase is the initiating phase, during which the mobile device sends Transfer-initiating messages to one of the Context Information Hosts located in its home network. The third phase, i.e. the context information transfer phase, is started by the home Context Information Host. During the third phase, context information is communicated between two remotely located Context Information Hosts.

Context information is most likely required by some enterprise applications and involved with allocation of valuable network resources. Consequently, security is very important in the process of global context information transfer. The general security policy for global context information transfer includes data integrity, data confidentiality and access control.

Data integrity ensures that what the receiving party receives is exactly the same as what the sending party sends. Data integrity is the most fundamental requirement for communicating data across public networks. There are two schemes to ensure data security. The first one is based on secret key cryptography [46] [47]. In this kind of cryptography, a secret key is established and shared between communicating peers, and the same key is used to encrypt and decrypt messages on both sides. Under the secret key scheme, the sending party encrypts the plaintext message using the secret key. It can reasonably be assumed, with sound encryption algorithm such as triple-DES, and a decent key length (512 – 1024), that the encrypted message cannot be decrypted without knowledge of the secret key. Consequently, data integrity is ensured because the

encrypted message can't be tampered with without being detected by the receiving party. Data confidentiality is also provided when a plaintext is encrypted. The secret key, which is distributed in a very secured manner, e.g. manually, also shows the identity of the message sender and it can be used for access control.

The second scheme is based on public key cryptography [46] [47]. A public key cryptography is a cryptosystem in which a user has a pair of mathematically related keys:

1. A public key that can be known by everyone
2. A private key that is assumed to never leave its owner

A more detailed description of the public key cryptography is available in [46] [47]. Briefly, a message sender provides data integrity in a public key cryptosystem by generating a digital signature, using the private key of one public-private key pair. A digital signature is a string of characters that provides an electronic analog of handwritten signatures. Any receiver can verify whether the digital signature matches the received message, as long as it knows the corresponding public key. To provide data confidentiality, the message sender encrypts the message using the receiver's public key and the encrypted message can only be decrypted by the intended receiver using the matching private key.

Inseparable from the public key cryptography is PKI, which stands for public key infrastructure. PKI was reviewed briefly in section 5.4.3.2.

The three phases of global context information transfer have different security requirements. The first phase, as described in section 6.2, is implemented using IETF's Service Location Protocol (SLP). The scope of SLP operations is limited to a local

network, where security intrusions are relatively easier to detect and prevent. Currently, the SLP standard does not require security measurements, and this project does not propose any security enhancement either. However, in the second phase, the messages exchanged between mobile devices and their home Context Information Hosts have to travel across public networks, which makes data integrity and data authentication fundamental requirements. Considering that a mobile device and its home Context Information Host are “unknown” parties, data encryption based on the private key cryptography, i.e. Digital Encryption Standard (DES) algorithm [46], is adopted in the second phase. In contrast, data communication in the third phase happens mainly between “unknown” parties, and protocols based on the public key cryptography and PKI must be relied upon.

The rest of this chapter describes in detail how the three phases shown in Figure 6.1 were implemented by this project. Section 6.2 describes the service discovery phase, followed by the description of the initiating phase in section 6.3. The context information transfer phase is described in section 6.4.

6.2 Service Discovery phase

6.2.1 Overview of the Service Location Protocol

Before a mobile device begins running applications that need context information in a foreign network, it first has to discover the Context Information Hosts in the area being visited. We selected the Service Location Protocol [48], SLP, as the protocol for finding foreign Context Information Hosts in the service discovery phase.

SLP is an IETF standard that provides a scalable framework for discovering local network services. SLP is decentralized, lightweight and extensible. It has been accepted by developers as a simple, minimum-requirement service discovery protocol [49].

According to reference [48], client applications in the SLP framework are modeled as the “User Agents” and services are modeled as the “Service Agents”. A third party, the “Directory Agent”, caches and concentrates service information for the local subnet. Communications between User Agents and Service / Directory Agents are mostly based on UDP multicast. According to the SLP specification, all Service Agents and Directory Agents are in the local “Administratively Scoped SLP Multicast Group”, the address of which is “239.255.255.253” with the port number 427 [48].

A User Agent gets service information either from the Service Agents or a Directory Agent. In the first case, a User Agent multicasts a service request message to the address “239.255.255.253”. A Service Agent receiving a request for a service which it advertises must unicast a reply containing the service’s location information. Directory Agents are used in SLP to decrease the amount of local traffic. If a User Agent knows any Directory Agent, it sends unicast service request messages to the Directory Agent. All the Directory Agents in a local network can be discovered using SLP, as well as other resource servers.

A network service in SLP is encoded as a service URL, which has the following format:

“service:”<service-type> “:/" <address-spec>

“<address-spec>” is usually a host name or a IP address in dotted form, followed by an optional *“:”* and a port number.

“<service-type>” contains information on service types, which has the following format:

<abstract-type> “:” <concrete-type>

The relationship between an “<abstract-type>” and “<concrete-type>” is analogous to that of interface and implementations in Java. For example, we use the term “Context-Information-Host” as abstract type for all hosts that are able to communicate context information, and the term “Context-Information-Host-SSL” indicating the hosts that can communicate context information using secured sockets connections [48].

The simplicity of SLP makes it suitable to be implemented in simple devices with small memory. Section 6.2.2 describes how this project implemented the User Agent part of SLP in J2ME, a platform portable for most handheld devices.

6.2.2 Prototyping of SLP user agent by this project

Java™ 2 Micro Edition, J2ME, is the Java™ 2 platform targeted at consumer electronics and embedded devices [50]. The J2ME architecture is composed of three layers: 1) the Java Virtual Machines layer, which is customized for a particular device’s operating system; 2) the Configuration layer, which defines a set of core Java class libraries; and 3) the Profile layer, which defines the minimum set of APIs targeted at a particular group of devices. The Configuration Layer for small, mobile devices is called the Connected, Limited Device Configuration (CLDC). CLDC can be deployed to devices such as mobile phones and simple PDAs on top of the Kilobyte Virtual Machine (KVM). Currently, the only Profile Layer available for CLDC is a Mobile Device

Information Profile (MIDP) that provides API classes for user interface, networking, persistence and application models.

All MIDP applications are subclasses of “*javax.microedition.midlet.MIDlet*”. After the source codes of MIDP applications have been compiled into bytecodes, they have to be pre-verified, a phase for simplifying the final stage of bytecode verification on the CLDC virtual machine. The pre-verified classes, together with resource files such as image files and text files, are packaged into a MIDlet suite that can be installed in any devices as long as CLDC and MIDP are deployed. J2ME provides a portable platform for developing applications targeted to mobile devices, and we have implemented the SLP User Agent part using CLDC and MIDP.

The SLP User Agent was implemented by the `getGateWay()` method in the class `SLPRequest`, the UML class diagram of which is shown in Figure 6.2.

Figure 6.2: The class diagram for implementing SLP user agent

SLPRequest is a subclass of J2ME version's "java.lang.Thread" so that service discovery can occur as a background process. The service request messages are multicasted through a UDP socket, i.e. the UDP_socket class shown in Figure 6.2, the type of which is "javax.microedition.io.DatagramConnection".

The service discovery process is implemented using two threads: the sending thread and the receiving thread. The sending thread is implemented using ResendTimer. As shown in Figure 6.2, the type of ResendTimer is "java.util.Timer". Each Timer object is associated with one or several tasks. Corresponding to each Timer object, the java virtual machine creates a background thread, which executes the tasks at the time specified by users. The task for the "resendTimer" is described using the following pseudo-code:

```
if current_time – begin_time > maximum_SLP_time then  
    close UDP_socket  
    terminate  
else then  
    send a SLP service  
    set resendTimer for next task
```

The receiving thread loops on receiving data from the UDP socket, until the sending thread closes the UDP socket and causes an IOException.

The above service discovery process requires many parameters, such as the maximum_SLP_time. It is undesirable to hard-code these parameters; however, according to CLDC and MIDP specification, different MIDlet suites cannot share data.

Consequently, it is not possible to write the parameters in a “shared database” or “system file”. Each MIDlet application that uses SLPRequest class has to include a configuration file with the name “thisfile.txt”. We provide the default “thisfile.txt” and programmers can change its settings. When a MIDlet creates a new SLPRequest instance, it passes itself as a reference to the SLPRequest instance (the “caller” class in Figure 6.2) so that all the resource files, including “thisfile.txt”, are also passed.

This project downloaded J2ME wireless toolkit 2.0 beta 2 (WTK20) from the following web page: <http://java.sun.com/products/j2mewtoolkit>. WTK20 provides CLDC and MIDP reference implementation as well as an emulator environment for developing and testing CLDC/MIDP applications. WTK20 includes KToolBar, which is a minimal development environment with graphical user interfaces for compiling, packaging, pre-verifying and executing MIDP applications. Figure 6.3 shows the layout of the WTK20 directory. The “bin” directory shown in Figure 6.3 contains the executable files for emulators, KToolBars, and so on. CLDC and MIDP reference implementation is contained in the directory “lib”. Applications developed by users are put in the “apps” directory. A subdirectory of “apps” is “lib”, where library classes developed by users are put. The SLPRequest class was developed using WTK2.0 and it has been put into the “apps/lib” directory so that it can be used by multiple applications.

Figure 6.3: WTK20 directory layout

6.3 Context information transfer-initiating phase

6.3.1 Message description

After a mobile handheld device gets access information about the foreign Context Information Hosts, it will make contact with one of its home Context Information Hosts by sending a Transfer-initiating message. The Transfer-initiating message is encrypted to ensure data integrity and authentication. The format of the Transfer-initiating message is shown in Figure 6.4, and the plaintext format of the Encrypted message field in Figure 6.5.

Figure 6.4: The format for Transfer initiating message

The MLength field is a short integer (2 bytes long) indicating the total length of the message. The SPI stands for Security Parameter Index. In order to communicate with its home Context Information Host, a mobile handheld device should normally be configured with a set of secret keys. The purpose of the SPI field is to uniquely identify which shared secret key is used to encrypt the following message payload. The identity of the mobile node is indicated in the HostName field. The value of HostName can either be

a domain name or an IP address in the dotted format. The length of the HostName is indicated in the HostLength field (1 byte long).

The plaintext format of the encrypted payload for the Transfer-initiating message is depicted in Figure 6.5.

Figure 6.5: The plaintext format for of the encrypted payload in the Transfer-initiating message

The AppLength field in Figure 6.5 is equal to the length of the AppName field. AppName is the name of the application for which context information is going to be transferred. The Current time field is set to the time when the mobile device composes the message.

The AccessInfo field contains contact information of the foreign Context Information Hosts. The AccessInfo field is copied from the service reply message in the SLP protocol, which is described in the previous section. The length of the AccessInfo field is indicated in the infoLength field.

After the home Context Information has accepted and processed (described in 6.3.2) the Transfer-initiating message, it replies with an acknowledgement message. The format is shown in Figure 6.6.

The MLength field and the SPI field have the same meaning as in Figure 6.4. The acknowledgement message is encrypted and the encryption key must be different from that used in the initiating message. The payload of the acknowledgement message is shown in Figure 6.7.

Figure 6.6: The format for Transfer-Initiating Acknowledgement Message

Figure 6.7: The plaintext content for of the encrypted payload in the Transfer Initiating Acknowledgement message

Values in the Current time field, the APPLength field and the AppName field are copied from the Transfer initiating message. The Code field indicates whether the request has been accepted or rejected. Section 6.3.2 describes how the Code field is set.

6.3.2 Functional description

6.3.2.1 Functional description for the home Context Information Host

When the home Context Information Host receives a Transfer-initiating message, it will first find the secret encryption key using values contained in the SPI field and the HostName field. If no secret key is found or the secret key fails to decrypt the payload, then the Context Information Host discards the Transfer-initiating message and terminates processing. Otherwise, the decryption is successful, and the home Context Information Host reads the value in the Current time field. For every served mobile device, a home Context Information Host must keep a “device-time” binding which has the following three fields:

1. Device-ID: uniquely identifies the served mobile device
2. Last-Request-Time: uniquely identifies when the mobile device sent the last Transfer-initiating message.
3. Last response message: the acknowledgement message for the last Transfer-initiating message.

The home Context Information Host compares the Current time field contained in the Transfer-initiating message and the Last-Request-Time in the “device-time” binding. If the Current time in the message is earlier than the Last-Request-Time, then the Transfer-initiating message should be discarded and the processing terminates. If the Current time is equal to the Last-Request-Time, then the last response message stored in the “device-time” binding is replied. Otherwise, the Current time is greater than the Last-Request-Time, which implies that the coming Transfer-initiating message is a new one

and the Home Context Information Host parses the AppName field and fetches the required context information. There are two situations:

Under the first situation, no context information is found. The home Context Information Host then makes an acknowledgement message and sets the Code field to be “0”, which indicates the request has been rejected due to an incorrect AppName. The home Context Information Host sends out the message. The home Context Information Host also updates the “device-time” binding for the mobile device by copying the Current time field value to the Last-Request-Time, and replacing the last response message with the newly created one.

Under the second situation, the context information is found. Then the home Context Information Host has to parse the value contained in AccessInfo field. If the home Context Information Host finds that the remote Context Information Host cannot be reached – for example, the remote Context Information Host requires SSL connection, while an SSL connection is not supported in the home Context Information Host – then an acknowledgement message with the Code field set to “2” is sent. “2” indicates the request is rejected due to impossible communication. Otherwise, the home Context Information Host accepts the request and the Code field in the acknowledgement message should be “1”. In both cases, the “device-time” binding is updated.

6.3.2.2 Functional description for the Mobile device

After a mobile device has sent out a Transfer-initiating message, it caches the Transfer-initiating message and waits for n seconds (n is configurable). If an acknowledgement message with a matching “Current time” field is received in n seconds,

the second phase is terminated. Otherwise, the mobile devices continue sending the same message every n seconds until an expected acknowledgement message is received, or a total of m (m , a configurable value) seconds has passed since the first initiating message.

The mobile device has to keep a global variable, i.e. “Current time”. The value of “Current time” is not necessarily equal to the current time; however, it always has to increase. A mobile device can set the “Current time” value by synchronizing with a time server in its home network, or by using Internet Time Services.

6.3.3 Prototyping by this project

6.3.3.1 Implementation in the home Context Information Host

Figure 6.8 shows the classes that were implemented for the home Context Information Host for processing Transfer-initiating messages.

TI_Msg_Processor class in Figure 6.8 implements the Transfer-initiating message processing logics, described in section 6.3.2.1. TI_Msg_Processor provides the following public method:

byte[] getAckMsg(byte[] in_msg)

The in-parameter of getAckMsg is the Transfer-initiating message sent by a mobile device with the “MLength” field excluded. The return value of getAckMsg can be a byte array containing the acknowledgement message, or a null value if the Transfer-initiating message is discarded.

The TI_Msg_Processor retrieves the secret key by calling the following method provided by SPI_Checker class:

byte[] getKey(byte[] SPI, byte[] mobile_device)

The SPI-Checker class is an abstract class which only defines the signature of the getKey method. As shown in Figure 6.8, SPI_Checker_Imp, a subclass of SPI_Checker, has been implemented in our prototyping. SPI_Checker_Imp reads the key information from a text file. The content of the text file is a repetition of the following line:

“Mobile_device=XXXX:SPI=XXXX:Key=XXXX”

SPI_Checker_Imp stores all secret keys for mobile devices in memory as an instance of java.util.HashMap, i.e. H1 in Figure 6.9.

Figure 6.8: Class diagram for the classes implemented in the home Context Information Host

Figure 6.9: Two hash maps are used to hold secret keys for all mobile devices

Entries in H1 are indexed by mobile device IDs. Given the value of one mobile device ID, theoretically H1 can return the corresponding HashMap instance (H2 in Figure 6.9) in constant time. As shown in Figure 6.9, entries of H2 are indexed by SPI values and contain secret keys for one mobile device.

The TI_Msg_Processor class uses a Cipher class to do the encryption and decryption. The Cipher class is implemented using classes in the “javax.crypto” package, a package included in J2SDK, v1.4. DES encryption was used in prototyping.

Another class needed by TI_Msg_Processor class is the Binder class, which maintains the “device_time” binding described in section 6.3.2.1. A Binder class is an abstract class which defines the signature for the following three methods:

synchronized int validTime(byte[] time, byte[] mobile_device)

synchronized byte[] getLastMsg(byte[] mobile_device)

synchronized void upateBinding(byte[] time, byte[] msg, byte[] mobile_device)

The Method validTime() compares the in-parameter “time” with the corresponding “Last_Request_Time”. The return values of method validTime() can be:

1. 0: when the in-parameter “time” is equal to “Last_Request_Time”
2. 1: when the in-parameter “time” is greater than “Last_Request_Time”
3. -1: when the in-parameter “time” is less than “Last_Request_Time”

The Method getLastMsg() returns the last message sent to the mobile device. And the method updateBind() updates the internal binding.

All the above methods are synchronized. When a thread calls a synchronized method, it must first get a system lock to lock the method. No other thread can call the same method before it is unlocked.

Figure 6.8 shows that class Binder is a singleton, i.e. only one instance is used in the whole application.

In our prototype, one “device-time” binding is stored in an instance of the class OneBinding. All the “device-time” bindings are maintained by the class Binder_imp, a subclass of class Binder. Class Binder_imp stores information using an instance of java.util.HashMap.

The Abstract class Info_Checker and its subclasses encapsulate the mechanism as to how a home Context Information Host decides whether required context information exists when processing a Transfer-initiating message. Class Info_Checker defines the following method:

```
bool info_available(byte[] appName)
```

In reality, the implementation of the method info_available() must rely on system configuration files, or even databases. And the subclasses of Info_Checker can be very complicated. However, our prototyping used class Info_Checker_imp, the most simplified implementation of class Info_Checker. In class Info_Checker_imp, the availability of information for mobile devices is hard-coded.

6.3.3.2 Implementation in mobile devices

For mobile devices, two classes – class Encryptor and class CT_Initiator – were implemented using CLDC and MIDP APIs.

Class Encryptor is a utility class and it defines the two following methods:

```
byte[] encrypt(byte[] msg, byte[] keyVal);
```

```
byte[] decrypt(byte[] msg, byte[] keyVal);
```

Method `encrypt()` does DES encryption for the in-parameter `msg`, using in-parameter `keyVal` as the encrypting key, while method `decrypt()` decrypts the in-parameter `msg` using `keyVal`.

Figure 6.10: Class `Encrptor` uses packages developed by Bouncy Castle

Data encryption is not a standard feature for either the Connected Device Configuration (CDC) or Connected Limited Device Configuration (CLDC) yet. Fortunately, there is an open-source Java encryption project hosted at “<http://www.bouncycastle.org>”, which has adapted some of its code to work with the mobile devices. Bouncy Castle has developed a set of lightweight Java API which can be installed in the J2ME platform and supports most of the common encryption algorithms including DES, blowfish and RC4. Class `Encrptor` was implemented using the lightweight cryptography API downloaded from Bouncy Castle, as shown in Figure 6.10.

The behavior of the mobile terminal in the second phase is implemented in the CT_Init() method of Class CT_Initiator. The CT_Init() method has the following method signature:

int CT_Init(byte[] msg, String url, int m, int n)

In-parameter msg in method CT_Init is the data to be sent out by the mobile device, while in-parameter url has the following format:

socket://domain name:port number

The meaning of in-parameter m and in-parameter n is described in the following paragraph.

The CT_Init() method first creates a TCP socket connection using the in-parameter url, and sends the first Transfer-initiating message. The main thread then waits for the acknowledgement message from the home Context Information Host, and another thread, the re-sending thread, re-sends the transfer_init message every n (n is equal to the in-parameter n) seconds. CT_Init() method is terminated in one of the following two manners:

1. The main thread closes the socket connection after it has received an acknowledgement message. The re-sending thread detects that the socket is closed when it tries to re-send data, and the re-sending thread also terminates.
2. The re-send thread detects that m (m is equal to in-parameter m) Transfer_initiating messages have been sent out, and the re-send threads closes the socket connection, which causes an IOException in the main thread. The main thread detects the closure of the socket, and terminates.

If no acknowledgement is received, the CT_Init() method replies -1. Otherwise, the value of the Code field in the acknowledgement message is returned.

As mentioned in the previous section, in this project we developed all the J2ME codes using the J2ME wireless tool kit. The downloaded Bouncy Castle cryptography API was put in the “app/lib” directory, which can be accessed by all the installed applications (refer to figure 6.3).

6. 4 Context information transfer phase

Option 1

Option 2

Figure 6.11: Two options to locate Context Information Protocol

This section describes the third phase of global context information transfer, i.e., communicating context information securely across public networks. Exchanging context information is carried out by the Context Information Protocol, which is a session-based, application-level protocol defined by this project. Figure 6.11 shows two options for locating the Context Information Protocol.

With the first option, the two hosts communicating the context information are connected using the secured socket layer (SSL) protocol and the Context Information Protocol is layered on SSL. As shown in Figure 6.11, the home Context Information Host is at the client side of the SSL socket connection, while the foreign Context Information Host is at the server side of the SSL socket connection.

With the second option, the Context Information Protocol is layered on the CT-Security Layer, which is a simple security protocol defined by this project. As shown in Figure 6.11, the CT-Security Layer is built directly on top of a TCP connection, with the home Context Information Host at the client side and the foreign Context Information Host at the server side.

The foreign Context Information Host advertises its services by sending one or both of the following messages:

1. *Context-Information-Host:Context-Information-Host-SSL://IP
address:port number*
2. *Context-Information-Host:Context-Information-Host-CT-Security-
Layer://IP address:port number*

“Context-Information-Host-SSL” indicates the SSL connection is supported in the advertised host, while “Context-Information-Host-CT-Security-Layer” indicates that CT-Security-Layer connection is to be used to access the service. The home Context Information Host chooses a common protocol that is supported at both sides. When an SSL connection and CT-Security-Layer connection are both available, SSL should be used.

SSL, the Context Information Protocol and the CT-Security-Layer are discussed in sections 6.4.1, 6.4.2 and 6.4.3 respectively.

6.4.1 The Secure Socket Layer protocol

6.4.1.1 Review of the Secure Socket Layer protocol

The Secure Socket Layer (SSL) protocol [51] was originally developed by Netscape Corporation as a security enhancement for their web servers and browsers. Since then, SSL has been widely deployed in many intranets as well as over the public networks, and it has become the *de facto* standard for transport layer security. SSL standardization is now under the control of the IETF, and SSL has been renamed Transport Layer Security (TLS) [52].

SSL connection is built on top of TCP. Data transport through SSL involves two phases. First, the two engaged parties optionally authenticate each other and then exchange session keys. The first phase is also known as the handshake phase. Once the first phase is over, the two parties encrypt data using the shared secret key and the encrypted data are transported using the underlying TCP layer, and begin the second phase [51].

Figure 6.12: Message sequence diagram for SSL protocol

SSL is intrinsically asymmetric and differentiates between a client and a server. The messages exchanged between a SSL client and an SSL server are illustrated in Figure 6.12.

The SSL messages are sent in the following order:

1. Client hello: The client hello message contains information on cryptographic algorithms and key sizes supported at the client side

2. Server hello: The server hello message contains the selected cryptographic algorithm and key size for setting up the SSL session
3. Certificate: The server sends the client a public key certificate or a certificate chain
4. Certificate request: The server sends a certificate request message if it also needs to verify the identity of the client
5. Server key exchange: The server sends the client a server key exchange message when the public key information sent in 3) is not sufficient for the key exchange
6. Server hello done: The server tells the client that it is finished with its initial cipher suit negotiation
7. Certificate: This is sent if the server requests a certificate from the client.
8. Client key exchange: The client generates and sends information used to create the session key
9. Certificate verify: This message is sent only when a client certificate is needed by the server. The purpose of this message is to allow the server to complete the process of authenticating the client
10. Change cipher spec: The client sends a message telling the server to change to encrypted mode
11. Finished: The client tells the server that it is ready for secured data communication
12. Change cipher spec: The server sends a message telling the client to switch to encrypted mode

13. Finished: The server tells the client that it is ready for secured data communication

14. Encrypted data: Data is communicated in encryption format

SSL provides a secured method of client-server communication, and has been used in many e-commerce applications. SSL is obviously one choice for communicating context information.

6.4.1.2 Open source code and library

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and open source toolkit implementing the SSL v2/v3 and Transport Layer Security (TLS v1) protocols. The SSL library for both Windows and Unix platforms can be downloaded from OpenSSL's home page, which is at "<http://www.openssl.org/source/>".

The prototype in this project made use of the Java™ Secure Socket Extension (JSSE), a set of Java packages (including "javax.net.ssl") that enable secure Internet communications. JSSE implements a Java version of SSL and TLS protocols and includes functionality for data encryption, server authentication, message integrity, and optional client authentication. JSSE has already been integrated into J2SDK, v1.4.

6.4.2. The Context Information Protocol

6.4.2.1 Message description

The Context Information Protocol covers the process in which a foreign Context Information Host receives context information from a mobile device's home Context Information Host. It is a session-based protocol and the messages exchanged in one session are shown in Figure 6.13.

Figure 6.13: The messages exchanged in the Context Information Protocol

The format for the messages exchanged in the Context Information Protocol is shown in Figure 6.14. MLength field contains the length of the message. The value in the Type field can be:

- 1: for the Meta-data message
- 2: for the positive Meta-data reply message

- 0: for the negative Meta-data reply message
- 4: for the no-effecting Meta-data reply message
- 3: for the Information message
- 5: for the Information acknowledgement message

Figure 6.14: The format for the messages exchanged in the Context Information Protocol

The values in both the H Session Number field and the F Session Number field are 4-byte unsigned integers. When the home Context Information Host composes the Meta-data message, it generates a new session number and puts it into the H session Number field. In the Meta-data message, the F Session Number field is unset. When the foreign Context Information Host receives a Meta-data message, it begins a new session and generates a 4-byte unsigned session number, the value of which is copied to the F Session Number field in the Meta-data reply message. The values in the H Session Number field and the F Session Number field are used for session tracking and stay unchanged throughout the rest of the session.

The home Context Information Host fetches all the required context information after it has accepted a Transfer-initiating request. Different sets of context information

may exist to serve one application. For example, either the DiffServ information set or the IntServ information set can be used for a videoconference application. Which set of information is to be used depends on the visited network. Each set of context information is described by a meta-data schema such as an XML schema file, or an element that can uniquely identify the meta-data schema, e.g. a namespace or a UDDI registry URL plus a tModelKey, as described in section 5.3.1. The home Context Information Host describes what context information it can transfer in the payload of the Meta-data message. The format of the payload is depicted in Figure 6.15.

Figure 6.15: The payload for the Meta-data message

The payload of the Meta-data message contains meta-data and information about the mobile device that requests context information transfer. The Group Number field in Figure 6.15 is a 2-byte unsigned integer indicating how many M groups are in the message. The Meta-data Groups field contains one or multiple M Groups.

Figure 6.15 shows that one M Group begins with the Type field. The value in the Type field can be:

- 1: The following “Content” field is an UDDI public registry URL plus a tModelKey
- 2: The following “Content” field is a *Transfer Data Definition* document
- 3: The following “Content” field is a name space URL

The Length field in one M group indicates the length of the Content field.

In addition to the M groups, the Meta-data message also contains:

1. The identity of the mobile device that requested the information transfer is in the Mobile Host field, the length of which is in the MHostLength field.
2. The identity of the application for which the information is transferred is in the AppName field, the length of which is in the AppLength field.
3. The Current time field is copied from the Transfer-initiating message (Figure 6.4).

The foreign Context Information Host processes the Meta-data message in the following manner:

The foreign Context Information Host checks whether any of the meta-data contained in the message is supported. If no meta-data is supported, then a negative Meta-data reply message, i.e. a Meta-data reply message with the “Type” field set to 0, is returned. Otherwise, the foreign Context Information Host checks whether the Meta-data request is “redundant”, i.e. whether context information for the same mobile device and

the same application has already been received. If this is the case, then a no-effecting Meta-data reply message is returned; i.e. the type field is 4. Otherwise, the foreign Context Information Host replies with a positive Meta-data reply message. The negative Meta-data reply message and the no-effecting Meta-data reply message have no payload, while the positive Meta-data reply message has the selected M group as its payload.

The payload of the Information message is decided by the matching meta-data schema, e.g. the payload can be a XML file, or even an ASN.1 BER file.

The purpose of the Information acknowledgement message is to acknowledge the receipt of the Information message, and it has no payload.

6.4.2.2 Functional description

The behavior of the home Context Information Host can be described using the state diagram shown in Figure 6.16.

The home Context Information Host is in the Meta-data state after sending out a Meta-data message. A positive Meta-data reply message brings the home Context Information Host into the Information state, in which an Information message is sent. If an Information reply message is received before time-out occurs, then the Success state is entered.

The home Context Information Host can transit from the Meta-data state directly to the Success state if a no-effecting Meta-data reply message is received.

A negative Meta-data reply message triggers the home Context Information Host into the Fail state.

Figure 6.16: State diagram for the home Context Information Host

If no Meta-data reply message is received in the Meta-data state in $n1$ seconds ($n1$ is configurable), or if no Information acknowledgement message is received in the Information state in $n2$ seconds ($n2$ is configurable), then the Retry state is entered.

In the Retry state, the home Context Information Host closes the current session and waits for $n3$ seconds ($n3$ is configurable) to begin a new session by sending a Meta-data message.

The home Context Information Host gives up after entering the Retry state $n+1$ times (n is configurable).

Figure 6.17 describes the behavior of the foreign Context Information Host.

Figure 6.17: The state diagram for the foreign Context Information Host

After a Meta-data message has been received, the foreign Context Information Host enters the Information State. If a positive Meta-data reply message is composed for the request, then the foreign Context Information Host stays in the Information state for m seconds (m is configurable). An Information message triggers the foreign Context Information Host to enter the Accept state. In the Accept state, the context information will be processed by the respective network elements.

If the foreign Context Information Host returns a negative or no-effecting Meta-data reply message, then the Reject state is entered. In the Reject field, the current session is closed. If no Information message is received after m seconds in the Information state, then the foreign Context Information Host enters the Reject state.

6.4.2.3 Prototyping in this project

6.4.2.3.1 Classes installed in the home Context Information Host

Figure 6.18 shows the classes for the sender in global context information transfer. These classes were installed in the home Context Information Host in prototyping by this project.

The state transitions of the home Context Information Host described in 6.4.2.2 are implemented by Sender and Re_Starter, which call each other recursively. Both Sender and Re_Starter are subclasses of java.lang.Thread so that they can be used to start new threads.

Figure 6.18: Classes diagram for the sender in global context information transfer

The following methods are provided by the class Sender:

```

Sender(String address, short port, byte[] mobile_device, byte[] appName, int
retry_num)
void run()

```

void send()

Method `Sender()` is the constructor method for class `Sender`. The in-parameter address and port are for the IP address and the port number of the remote host. The in-parameter `mobile_device` and `appName` identify the mobile device and application for which the context information is to be transferred. When a `Sender` instance is first created, the value for in-parameter `retry_num` is 0. `retry_num` is equal to the number of times the home Context Information Host tries to send data to the foreign Context Information Host. How `retry_num` is updated will be discussed later.

Method `run()` is inherited from `java.lang.Thread`, and it works by calling the method `send()`.

The pseudo-code of method `send()` is listed as Figure 6-19:

1. Send out a Meta-data message
2. Create a new Restarter instance `r1` (`r1 = new Re_Starter(socket, address, portNum, mobile_device, appName, retry_num + 1)`)
3. Set timer `t1` to execute the `run()` method of `r1`, `n1` seconds later.
4. Wait for the Meta-data reply message
5. If a Meta-data reply message is received, then
 6. Cancel `t1`
 7. Compose and send out the Information message
 8. Set timer `t2` to execute the `run()` method of `r1`, `n2` seconds later.
 9. Wait for Information acknowledgement message
 10. If an Information acknowledgement message is received, then
 11. Cancel `t2`
 12. Return

Figure 6.19: The pseudo code for the method `send()` in `Sender` class

Before the `send()` method begins to wait for the Meta-data reply message and the Information acknowledgement message, it sets a timer which will time-out `n1 / n2` seconds later (line 3 and line 8 in Figure 6-19). When the timer times-out, the `run()`

method of Re_Starter is executed in another thread. The run() method of Re_Starter can be summarized as:

1. Close the socket
2. If `retry_num < m` then
 - a. Create a new instance of Sender
 - b. Set a timer `t2` which will time-out `n3` seconds later, and when `t2` times out, the run method of Sender is to be executed.
3. Else terminate

Re_Starter closes the socket and forces the thread which waits for reply messages to terminate. Re_Starter compares the value of `retry_num` with `m` (`m` is the maximal retry times) and if `retry_num` is less than `m`, Re_Starter will try to execute class Sender's `send()` method `n3` second later. Notice that `retry_num` is increased by one whenever a new Re_Starter instance is created (line 2 in Figure 6.19).

In addition to class Sender and class Re_Starter, Figure 6.18 also shows utility classes including MsgComposer, Info_Fetcher and Snum_Generator.

The MsgComposer class defines the following two methods to make the Meta_data and Information messages:

byte[] makeMetaMsg(byte[] mobile_device, byte[] appName)

byte[] makeInfoMsg(byte[] mobile_device, byte[] appName)

The operations of both the `makeMetaMsg()` method and `makeInfoMsg()` method depend on the class Info_Fetcher. Info_Fetcher is an abstract class. All subclasses of Info_Fetcher must implement the following two methods:

```
byte[] getMetaData(byte[] mobile_device, byte[] appName);
```

```
byte[] getContextInfo(byte[] mobile_device, byte[] metadata);
```

The implementation of method `getMetaData` and method `getContextInfo` can be very complicated, and may involve configuration files, databases or even some network protocols. In our prototype, class *Info_Fetcher_Imp* provides the most simplified implementation of class *Info_Fetcher*. *Info_Fetcher_Imp* hard-codes mobile devices / applications with a fixed set of context information data.

Another important utility class in Figure 6.18 is *Snum_Generator*. *Snum_Generator* provides the session number. It is a singleton, i.e. one instance is used in the whole application. *Snum_Generator* defines one method:

```
synchronized byte[] getSessionNum()
```

The `getSessionNum` method returns a unique session number, which is a 4-byte unsigned integer.

6.4.2.3.2 Classes installed in the foreign Context Information Host

Figure 6.20 shows the classes that were installed in the foreign Context Information Host.

The foreign Context Information Host's behavior, which is described in Figure 6.17, is implemented by the *Receiver* class and the *Terminator* class. The class *Receiver* implements the following public method:

```
void receive_process(Socket s)
```

The method `receive_process()` begins by passing the received *Meta_data* message to the class *ReplyMsgComposer*. *ReplyMsgComposer* returns a *Meta_data* reply message

to the Receiver class. The Receiver class sends out the reply message, and if the reply message is “negative” or “non-effective”, the `receive_process()` method terminates. Otherwise, the `receive_process()` method begins to wait for the Information message, which can be terminated by the Terminator if n seconds pass and no Information message is received.

Figure 6.20: Classes to be installed in the foreign Context Information Host

Figure 6.20 shows two abstract classes: the `Meta_data_Mapper` and the `Dispatcher` class. The subclasses of the `Meta_data_Mapper` implement the business logic, which is used to decide what context information for a given application is supported in

the local network, while the subclasses of the Dispatcher dispatch the received context information to the application that can interpret the information and make the corresponding configuration changes.

The subclasses of the Meta_data_mapper must implement the following method:

Meta_data_result getMeta_data_reply_msg (byte[] meta_data_msg)

The Method getMeta_data_reply_msg returns an instance of the class Meta_data_result. A Meta_data_result class contains the following public fields:

1. A byte array containing Meta_data reply message
2. An application name, i.e. a String which identifies the application which can understand the corresponding context information and make the corresponding configuration changes.

The Dispatcher is an interface which defines the method signature for the following method:

*void dispatch (byte[] information, byte[] mobileDevice,
String application);*

The in-parameter information and the in-parameter mobileDevice are used to determine the received context information and the mobile device identity, while the in-parameter application is the name of the application that knows how to process the context information, i.e. the rendering application depicted in Figure 4.4.

The subclass of the Dispatcher implements the way in which the rendering applications are invoked. Dispatcher_imp, the Dispatcher subclass implemented in this project, uses the Java Reflection APIs to dynamically invoke rendering applications. In

our prototype implementation, all the rendering applications have to implement a `RenderApp` interface with the following public method:

```
void process();
```

The rendering applications also have to provide a constructor that takes two byte arrays as in-parameters: one for context information, and one for the mobile device. `Dispatcher_imp` creates these rendering classes given their class passes at runtime. How the rendering applications process context information is not covered in this project.

6.4.3 The CT-Security Layer

6.4.3.1 Message description

The CT-Security Layer provides security services, including both data encryption and digital signature, to the context information protocol. It is similar to SSL but has been greatly simplified. In the CT-security layer, the generation of a session key is optional. According to [53], the session key generation brings most of the overhead in SSL. Avoiding generating a session key boosts performance greatly. The CT-security layer only supports RSA and DES. The messages exchanged between the home Context Information host and the foreign Context Information Host in the CT-security layer are shown in Figure 6.21.

All the messages passed through the CT-Security Layer have the same format shown in Figure 6.22.

Figure 6.21: The messages exchanged to set up the CT-Security Layer connection

Figure 6.22: Format for messages passing through CT-Security layer

The value in the MLength field is equal to the length of the message, including header and payload. The operations of the CT_Security Layer are session-oriented. The C Session Number field contains the session number generated in the client side (i.e. the home Context Information Host) and the S Session Number field contains the session number generated at the server side (i.e. the foreign Context Information Host).

The value of the Type field can be:

- 1: Encryption mode Client public key message
- 2: No encryption mode Client public key message
- 3: Encryption mode Server public key message
- 4: No encryption mode Server public key message
- 5: Encryption mode Client reply
- 6: No encryption mode Client reply
- 0: Data message, i.e. message containing data from the Context Information

Protocol layer.

The home Context Information Host sends the Client public key message to the foreign Context Information Host. If the home Context Information requires data encryption for the following Data messages, then the Type field in figure 6.22 is set to 1; otherwise the Type field is set to 2.

The payload of the Client public key message contains an X.509 public key certificate (CA field) as shown in Figure 6.23. The payload also contains a digital signature that covers the header portion shown in Figure 6.22.

Figure 6.23: Payload for the Client Public Key message

After the foreign Context Information Host has received the Client public key message, it verifies the public key using the attached certificate. If the public key is valid, then the foreign Context Information Host will also verify the digital signature. The

foreign context Information Host has to generate a session key under the following two conditions:

1. The foreign Context Information Host is required to use encryption, or
2. The Type field in the coming Client public key message is “1”.

If a session key is generated, the foreign Context Information Host encrypts the session key using the home Context Information Host’s public key, which has been received from the Client public key message, and replies with an encryption mode Server public key message. Otherwise, a no encryption mode Server public key message is sent back. The payload of the Server public key message is shown in Figure 6.24.

Figure 6. 24: Message format for the Server Public Key Message

The Encrypted Key field contains the encrypted session key, and its length is included in the “ELength” field. For a no encryption mode Server public key message, the value of the ELength field is 0. The Digital Signature field contains a digital signature which covers the message header portion, Elength field, and Encrypted Key field. The digital signature can be verified using the value in the CA field. The home Context Information Host verifies the Server public key message. If encryption mode is to be used, the home Context Information Host also decrypts the session key. The payload of the Client reply message only contains a digital signature, which covers the header portion of the message plus the server CA.

6.4.3.2 Functional description

Figure 6.25 describes the behavior for the home Context Information Host in the CT-Security Layer. After the home Context Information Host has sent out the Client public key message, it enters the ACK state, waiting for the Server public key message. If a valid Server public key message is received in $n1$ seconds ($n1$ is configurable), then the home Context Information Host transits to the Ready state, i.e., it is ready to encrypt data or make a digital signature for data coming from upper layer. If an invalid Server public key message is received, then the home Context Information Host enters into the Reject state. In the Reject state, the home Context Information Host closes the socket connection and writes an error message to the log file.

Figure 6.25: The state diagram for the home Context Information Host in the CT-Security Layer

The Retry state is entered if no message has been received after $n1$ seconds. In the Retry state, the home Context Information Host waits for $n2$ seconds, then re-sends the Client public key message and transits to the ACK state. However, if the Retry state has been entered consecutively m times, the home Context Information Host will transit from the Retry state to the Reject state.

The transition from the Ready state to the Closed state happens when the socket connection is closed by the upper layer, i.e. the Context Information Protocol layer. The Ready State transits to the Reject state if the upper layer has more data to send but it is detected that the other side has closed the socket connection.

The behavior of the foreign Context Information Host is shown in Figure 6.26.

Figure 6.26: The state diagram for the foreign Context Information Host

The foreign Context Information Host enters the ACK state after it has sent the Server public key message. Receiving a Client reply message within $n1$ seconds ($n1$ is a configurable parameter) causes the foreign Context Information Host transit to the Ready state; otherwise, the Reject state is entered.

If the foreign Context Information Host detects the socket connection has been closed by the other side in the Ready state, it transits to the Reject state. However, if the socket connection is closed by the upper layer, then the closed state is entered.

Figure 6.27: Class diagram for the CT_Security Layer classes installed in the home Context Information Host

6.4.3.3 Prototyping

Figure 6.27 shows the CT-Security Layer classes implemented for the home Context Information Host by this project.

The process of exchanging a public key / session key, as described above, is implemented in the constructor method of `SL_Sender`. The signature of the constructor method is:

```
SL_Sender(java.net.Socket socket, Public_Key_Man pubMan, Private_Key_Man priMan, int retryNum, Boolean encryption)
```

The in-parameter `socket` is a TCP socket that connects the home Context Information Host to the foreign Context Information Host. The in-parameter `pubMan` provides public key-related services, such as retrieving the public key and the public key certificate for the home Context Information Host, verifying the public key certificate sent by the foreign Context Information Host, making a digital signature and verifying digital signatures. The class `Public_Key_Man` is implemented using the classes from the following J2SDK packages:

`java.security.cert`: A package that provides classes and interfaces for parsing and managing certificates, certificate revocation lists (CRLs), and certification paths.

`java.security`: A package containing security classes, including that used for a digital signature.

The in-parameter `priMan` provides private key-related services such as generating a new session key, encryption, and decryption. The type of `priMan` is `Private_Key_Man`, which is implemented using classes from the following packages in J2SDK:

javax.crypto.cipher: A package provides classes and interfaces related to private key operations, including key generation, encryption and decryption.

Class SL_Sender provides the following four public methods:

VerifiedInputStream getVerifiedInputStream();

DecryptInputStream getDecryptedInputStream();

SignOutputStream getSignOutputStream();

EncryptOutputStream getEncryptOutputStream();

As shown in Figure 6.27, both the DecryptInputStream and VerifiedInputStream classes use a class with the type “java.io.InputStream” to read data from the socket connection.

Class DecryptInputStream provides the following public constructor:

DecryptInputStream(InputStream in, SecretKey sKey);

The in-parameter provides the InputStream from where data is read, while the in-parameter sKey is used to decrypt the encrypted payload.

String getPlainMsg();

The Method getPlainMsg() returns the message payload (refer to figure 6.22) in plain text. The implementation of getPlainMsg() is summarized using the following pseudo-codes:

- 1. Read three bytes from the underlying input stream and convert it to message length*
- 2. Read the whole message, i.e. read (message length – 3) bytes*
- 3. Decrypt the payload*

The constructor for class `VerifiedInputStream` is:

`VerifiedInputStream(InputStream in, X509Certificate cert).`

The In-parameter `cert` is the X.509 certificate sent by the other side, i.e. the foreign Context Information Host. The Class `VerifiedInputStream` also provides the method:

`String getPlainMsg();`

The operation of the `getPlainMsg()` method in class `VerifiedInputStream` is comparable to that in class `DecryptedInputStream`, though digital signature verification is undertaken rather than decryption.

Class `SignOutputStream` and class `EncryptOutputStream` are designed to send out data that has been properly signed or encrypted. Both `SignOutputStream` and `EncryptOutputStream` use `java.io.OutputStream` for sending data through the socket connection, and they have the following constructors:

`SignOutputStream(OutputStream out, Private pKey);`

`EncryptOutputStream(OutputStream out, SecretKey sKey);`

The `pKey` parameter in the `SignOutputStream` constructor is the private key used by the local host. Both the `SignOutputStream` and `EncryptOutputStream` classes also provide the following public method:

`boolean sendMessage(String message);`

The in-parameter `message` is the data sent by the context information protocol layer, which is encrypted or digitally signed by the `EncryptOutputStream` / `SignOutputStream` class.

Figure 6.28: Classes implemented in the foreign Context Information Host for the CT-Security Layer

Figure 6-28 shows the classes implemented for the foreign Context Information Host. Most of the classes shown in Figure 6-28 are the same as those shown in Figure 6-27, except class `SL_Receiver` and class `SL_Terminator`, which implement the behavior shown in Figure 6-26. Class `SL_Receiver` has the following Constructor:

```
SL_Receiver(java.net.Socket socket, Public_Key_Man pubMan,  
Private_Key_Man priMan, Boolean encryption );
```

The in-parameters have the same meanings as those for the SL_Sender constructor.

Similar to class SL_Sender, class SL_Receiver also has the following four public methods:

```
VerifiedInputStream getVerifiedInputStream();
```

```
DecryptInputStream getDecryptedInputStream();
```

```
SignOutputStream getSignOutputStream();
```

```
EncryptOutputStream getEncryptOutputStream();
```

6.5 Mobile device starting service-centric applications

After the foreign Context Information Host has received the context information, the mobile device should be informed that the service-centric application should start to run. There may be several ways for the foreign network to inform the mobile device, either via the IP layer connection or via the sub-IP layer connection. In our prototype, the foreign Context Information Host sends a UDP message to the mobile device.

6.6 Testing of prototype implementation

In order to test the prototyped classes described in the previous sections, the following scenario was implemented. Suppose a mobile user has just entered a visited

network and he plans to run a service-centric application, e.g. the videoconference application as mentioned in our motivating scenario. The scenario begins with his mobile device multicasting an SLP service request message. A dummy SLP server replies to the request by sending an SLP service message. The mobile device then sends a Transfer-initiating message to its home Context Information Host, e.g. router E0 in Ottawa site as described in our motivating scenario (Figure 4.2). The home Context Information Host obtains the access information of the foreign Context Information Host, e.g. router Ex in Toronto site as described in our motivating scenario (Figure 4.2), and transfers the context information by means of the Context Information Protocol. Once it has received a context information message, the foreign Context Information Host dispatches it to a rendering application and sends a “service ready” message, which is received by the mobile device.

Figure 6.29: Testing of the prototype classes

In order to implement the scenario, four applications were written and installed on the mobile device, home Context Information Host and foreign Context Information Host respectively.

M application: as shown in Figure 6.29, the M application is installed in mobile devices. M application calls the `getGateWay()` function of class `SLPRequest` (described in section 6.2.2) to get the address and access information for the foreign Context Information Hosts. The M application then calls the `CT_Init()` function of the class `CT-Initiator` (described in section 6.3.2) to communicate with its home Context Information Host. After a `CT_Init()` function has returned, the M application waits to receive the “service ready” signal which is to be multicasted by the F application.

SLP dummy server: This responds to a service request message by replying with a message that contains the foreign Context Information Host’s access information.

F application: The F application integrates all the classes described in Figure 6.20 and Figure 6.28 together, to provide the foreign Context Information Host functions. The F application opens two TCP servers: one is via a SSL server socket, and the other is via a normal TCP server socket. The two servers listen on different port numbers. If there is any client connection, the F application creates a new thread to handle the client request by calling the `receive_process()` method of the class `Receiver` (described in section 6.4.2.3.2). If the connection is via the normal TCP server socket, an instance of an `SL-Receiver` is also created to apply data protection. F application can be installed in visited networks, e.g., in Toronto site for our motivating scenario.

H application: the H application implements the functions described for a home Context Information Host. The H application opens a TCP server socket that waits for

connections from mobile devices. For each client connection, a new thread is created. The thread first processes a Transfer-initiating message by calling the `getAckMsg()` method of the `TI-Msg-Processor` class (described in section 6.3.3.1), then it connects to the F application using a SSL client socket or a SL-Sender. After the lower layer connection has finished, the thread begins the Context Information Protocol by calling the `run()` method of the `Sender` class (described in section 6.4.2.3.1). H application can be installed in home networks, i.e. the Ottawa network for our motivating scenario.

The M application was installed in the smart phone emulator provided by WTK2.0 (section 6.2.2). The H application was installed in one PC in our lab, while the F application and the SLP dummy server was installed in another PC in our lab.

Interactions among M application, SLP dummy server, H application and F application are described by Figure 6.30(a) and Figure 6.30(b), using UML message sequence diagrams, in which an instance of H application is named as “Ottawa”, while an instance of F application is named as “Toronto”. In figure 6.30(a), H application and F application communicates each other via CT_Security layer. The exchanged messages are described as follows.

1. SLP request: A SLP service request message sent by the M application which is installed in a mobile terminal
2. SLP reply (CT): Dummy SLP server replies the SLP request message by a service reply message which contains accessing information in the following format: “*Context-Information-Host:Context-Information-Host-CT-Security-Layer://IP address:port number*”.

3. Transfer initiate: M application sends a Transfer initiate message to H Application
4. Transfer initiate ack: H application process the received Transfer initiate message and sends a feedback message indicating whether the context information transfer request is accepted or not
5. Client public key: H application accepts the transfer request and communicates with F application via CT-Security layer. H application sends its public key and public key certificate to F application
6. Server public key: After it verifies the received Client public key message, F application sends its public key to H application
7. Client ack.: H application acknowledges its receiving of the Server public key message
8. Meta data: H application begins running Context Information Protocol by sending a Meta-data message
9. Meta data reply: F application replies the Meta data message sent by H application
10. Context info.: H application sends context information
11. Context info. ack.: F application acknowledges that a context information message has been received
12. Render info: F application passes the received context information to a render application. In or prototype, the rendering application just prints out the received information

13. Service ready: F application informs the mobile device that it can start running the service-centric application. In our prototype, a “start app” message is shown in mobile device after it has received this message

H application and F application communicate with each other via SSL in Figure 6.30(b). Messages shown in Figure 6.30(b) are almost identical to those in Figure 6.30(a), except that the service reply message advertises the following service message: “*Context-Information-Host:Context-Information-Host-SSL://IP address:port number*”, indicating the foreign Context information Host can be communicated using SSL socket.

Figure 6.30(a): The UML message sequence diagram describing the interactions among M application, F application, H application and SLP dummy server. H application and F application communicates via CT-Security layer

Figure 6.30(b): The UML message sequence diagram describing the interactions among M application, F application, H application and SLP dummy server. H application and F application communicates via SSL

Please notice that messages exchanged in SSL are omitted in Figure 6.30(b) for simplicity sake.

6.7 Summary

The purpose of this chapter was to show the feasibility of achieving global context information transfer. As mentioned in section 4.3, there are three scenarios in

which global context information transfer can be arranged, and this chapter has discussed in detail how the first scenario can be achieved in three phases.

The first phase, the discovery phase, can be implemented using the current Internet service-discovering facility. Although other protocols, such as Dynamic Host Configuration Protocol (DHCP), can also be applied, we chose SLP for this project because of its service-describing capability.

The design of the second phase, the transfer-initiating phase, has been influenced by Mobile IP. The mobile device and its home Context Information Host authenticate each other using secret key cryptography in a way similar to the registration authentication in Mobile IP [11].

Three protocols have been discussed for the third phase, the context information transfer phase. The Context Information Protocol is a session-based, application-level protocol designed by this project. It is used to pass context information from the home Context Information Host to the foreign Context Information Host. The Context Information Protocol does not provide any security protection and it has to be layered on top of the SSL, or the CT-Security layer when an SSL connection is not feasible.

This chapter has also described the prototyping implementation provided by this project for each phase. The prototyping shows that the operation of global context information transfer can be achieved using the current Internet architecture.

Chapter 7 Summary and future works

7.1 Summary of this project

Wireless networking is one of the driving forces for today's high-tech industry, while running applications for business professionals is one of the driving forces for the development of the wireless network. Opening a corporate network to mobile access is attractive to many enterprises with today's professionals spending more and more time working away from the office. As has been mentioned in previous sections, a very economic way of building a corporate network is to use IP level VPN, which enables running intranet applications using the Internet. The IP QoS and IP security management protocols make IP VPN a feasible solution today.

However, protocols such as the DiffServ protocol and the IPSec protocol are designed for fixed networks and require pre-configuration. The lack of mobility support in the "value-adding" Internet protocols can cause serious problems in achieving "anywhere, anytime, and any application" network service access, as has been described in Chapter 4. Deploying context information transfer globally provides a generic solution to these problems.

Global context information transfer is an inter-disciplinary topic, combining both engineering and management issues. In order to simplify the situation, we divided the Internet into multiple CIT domains. The operation inside one CIT domain becomes an engineering problem with the assumption that other issues, such as interoperability and

trust, can be solved by network management means. On the other hand, in the process of exchanging information between CIT domains, interoperability and trust issues are among the first which should be considered. This project has concentrated on how to solve interoperability and trust issues using XML, Web Service technologies, and public key cryptography technologies. Contributions from this project can be summarized as:

1. A proposal: this project has proposed that global context information transfer is one mechanism to set up network services for mobile users in visited networks
2. A framework: the Context Information Publish Framework described in Chapter 5 tries to solve the interoperability issues by guiding the developers on how to define and publish context information
3. A scenario implementation: how global context information transfer operates is documented in Chapter 6. The illustrations in chapter 6 show the feasibility of global context information transfer

7.2 Conclusions

The following conclusions are drawn from this project.

A mechanism for passing configuration data globally is required to take advantage of the full potential of today's Internet. Global context information transfer is one way of providing such a mechanism.

The dynamic and heterogeneous nature of the Internet makes it unfeasible to have one uniform format for all context information. Consequently, understanding context information is one of the challenges facing us. This project showed that both syntactic

and semantic understanding of context information can be achieved using XML and web service technologies.

Finally, this project prototyped an end-to-end solution for transferring context information securely across the Internet. The prototype was based on well-established technologies, including SLP and SSL, and it is concluded that global context information transfer is feasible with today's Internet.

7.2 Future work

This project had provided an end-to-end solution for transferring context information, however, the efficiency of the solution has not been tested yet. According to the proposed solution, context information can be transferred in XML format. Parsing XML is a relatively slow process. How to test and improve performance should be studied later.

The global context information transfer solution studied in Chapter 6 was at the application-layer level. Using IPv6 [54] technology, it is possible to build a solution in which context information is communicated at the network-layer level. Studying the pros and cons of these solutions is an interesting topic.

This project only addresses global context information transfer, while the local context information transfer, an inseparable part of the whole process, has not yet been covered. How to distribute context information efficiently in a local network, e.g. the IEEE 802.11 wireless LAN or the GPRS subnet in GSM, will complement the research done on this project.

How to render the context information into the network configuration is another important issue that is not covered in this project. Currently, the configurations for network elements are relatively stable, but context information transfer brings the problem of short-term configurations. The impact of context information transfer on network management and resource management should be investigated later.

References

1. Y.-B. Lin and I. Chlamtac, "Introduction" in *"Wireless and mobile network architecture"*, edited by C. Long, John Wiley & Sons, New York, 2001
2. Y.-B. Lin and I. Chlamtac, "GSM system overview" in *"Wireless and mobile network architecture"*, edited by C. Long, John Wiley & Sons, New York, 2001
3. J. Hämäläinen, "General packet radio service standard overview", in *"GSM, evolution towards 3rd generation systems"*, edited by Z. Zvonar, P. Jung, and K. Kammerlander, Kluwer, Academic publishers, Boston, 1999
4. R. Kalden, I. Merrick, and M. Meyer, "Wireless Internet access based on GPRS", in *IEEE personal communications*, page(s) 8 – 18, Apr., 2000
5. G. Patel and S. Dennett, "The 3GPP and 3GPP2 movements toward an all-IP mobile network", in *IEEE personal communications*, page(s) 62 – 64, Aug. 2000
6. G. Priggouris, S. Hadjiefthymiades, and L. Merakos, "Supporting IP QoS in the General Packet Radio Service", in *IEEE Network*, page(s) 8 – 17, Sept. / Oct. 2000
7. "Network reference model for cdma2000 spread spectrum system", revision A, *3GPP2 S.R005-A*, Dec. 1999
8. I. Guardini, P. Durso, and P. Fasano, "The role of Internet technology in future mobile data systems", in *IEEE communications magazine*, page(s) 68 – 72, Nov. 2000,
9. "Wireless IP network standard", *3GPP2 P.S001-A-1*, version 1.0, Dec. 1999

10. P. J. McCann, and T. Hiller, "An internet infrastructure for cellular CDMA networks using mobile IP", in *IEEE personal communications*, page(s) 26 – 32, Aug. 2000
11. C. Perkins, "IP mobility support for IPv4", *RFC 3220*, Jan. 2002
12. B. Gleen, et al, "A framework for IP based virtual private network", *RFC2764*, Feb. 2000
13. G. Mamais, et al, "Efficient buffer management and scheduling in a combined IntServ and DiffServ architecture: a performance study", *ICATM'99, 2nd International Conference on*, page(s) 236 –242, 1999
14. S. Shenker, C. Patridge, and R. Guerin, "Specification of guaranteed quality of service", *RFC 2212*, Sept. 1997
15. J. Wroclawski, "Specification of the controlled-load network element service", *RFC2211*, Sept. 1997
16. S. Shenker and J. Wroclawski, "General characterization parameters for integrated service network elements", *RFC 2215*, Sept. 1997
17. S. Blaker, et al, "An architecture for differentiated services", *RFC 2475*, Dec. 1998
18. J. Heinanen, et al, "Assured Forwarding PHB group", *RFC 2597*, June 1999
19. V. Jacobson, K. Nichols, and K. Poduri, "An Expedited Forwarding PHB", *RFC 2598*, Jun. 1999
20. D. Godens, et al, "Service level specification semantics and parameters", *Internet Draft*, Nov. 2000

21. S. Kent and R. Atkinson, "Security architecture for the internet protocol", *RFC 2471*, Nov. 1998
22. S. Kent and R. Atkinson, "IP authentication header", *RFC 2403*, Nov. 1998
23. S. Kent and R. Atkinson, "IP encapsulating security payload (ESP)", *RFC 2406*, Nov. 1998
24. J. Kempf, "Problem description: reasons for performing context transfers between nodes in an IP access network", *RFC 3374*, Sept. 2002
25. K. Le, et al, "Efficient and robust header compression for real-time services", *Wireless Communication and Networking Conference*, page(s) 924-928, 2000
26. C. Burmeister, et al, "Robust Header Compression (ROHC): framework and four profiles: RTP, UDP, ESP, and uncompressed", *Internet-Draft*, Feb. 2001
27. H. Syed, et al, "General requirements for context transfer", *Internet Draft*, Jan. 2002
28. W. Stallings, "IEEE 802.11: moving closer to practical wireless LANs", in *IEEE IT Professional*, page(s) 17-23, May-Jun. 2001
29. A. Yew, et al, "Quality of service management for the Virtual Home Environment", in *Proceedings of the 12th IEEE/IFIP International Workshop on Distributed Systems: Operations and Management (DSOM '01)*, page(s) 179-190, Oct. 2001
30. T. Bray, et al, "Extensible Markup Language (XML) 1.0 (second edition)", *W3C Recommendation*, Oct. 2000
31. D. C. Fallside, "XML Schema Part 0: Primer", *W3C Recommendation*, May 2001

32. D. Ehnebuske, D. Rogers, and C. von Riegen, "UDDI Version 2.0 Data Structure", *UDDI open draft specification*, June 2001
33. B. Mckee, D. Ehnebuske, and D. Rogers, "UDDI version 2.0 API specification", *UDDI open draft specification*, June 2001
34. T. Bellwood, et al, "UDDI version 3.0", *UDDI published specification*, Jul. 2002
35. N. Mitra, "SOAP version 1.2 part 0: primer", *W3C working draft*, Dec. 2001
36. H. Gomaa, "Use case modeling", in "*Designing concurrent, distributed, and real-time applications with UML*", Addison-Wesley, New York, 2000
37. D. Eastlake, XML-signature syntax and processing, *W3C recommendation*, February 2002
38. R. W. Younglove, "Public Key Infrastructure, how it works", in *Computing & Control Engineering Journal*, page(s): 99 – 102, April 2001
39. W. Ford and M. S. Baum, "Certificate" in "*Secure Electronic Commerce, Building the Infrastructure for Digital Signature and Encryption*", second edition, edited by A. Trowbridge, Prentice-Hall, Inc., 2001
40. P. Hallam-Baker, "XML key management specification", *W3C working draft*, Mar. 2002
41. "JavaTM Servlet Specification", version 2.4, *Public Draft from Sun Microsystems*, Jun. 2002
42. "JavaServer PagesTM Specification", version 2.0, *Proposed Final Draft from Sun Microsystems*, Aug. 2002

43. A. Chauhan, "The Client Tier" in *"Designing Enterprise Applications with the Java™ 2 platform, Enterprise Edition"* edited by N. Kassem and the enterprise team from Sun Microsystems, Oct. 2000
44. V. Khanna, "The Enterprise JavaBeans Tier" in *"Designing Enterprise Applications with the Java™ 2 platform, Enterprise Edition"* edited by N. Kassem and the enterprise team from Sun Microsystems, Oct. 2000
45. "Enterprise JavaBeans™ specification", version 2.0, *Specification Final Release from Sun Microsystems*, Aug. 2001
46. J. L. Massey, "An introduction to contemporary cryptography", in *Proceedings of the IEEE*, page(s): 533 – 549, May 1998
47. W. Ford and M. S. Baum, "Information Security Technologies" in *"Secure Electronic Commerce, Building the Infrastructure for Digital Signature and Encryption"*, second edition, edited by A. Trowbridge, Prentice-Hall, Inc., 2001
48. E. Guttman, et al., "Service Location Protocol", *RFC 2608*, Jun. 1999
49. S. Helal, "Standards for service discovery and delivery", in *IEEE Pervasive Computing*, page(s): 95 –100, Jul.-Sept. 2002
50. S. Helal, "Pervasive Java" in *IEEE Pervasive Computing*, page(s): 82 – 85, Jan.-Mar. 2002
51. W. Chou, "Inside SSL: the secure sockets layer protocol", in *IT Professional*, page(s): 47 – 52, Jul. – Aug. 2002
52. T. Dierks and C. Allen, "The TLS protocol", *RFC 2246*, Jan. 1999
53. G. Apostolopoulos, et al, "Securing electronic commerce: reducing the SSL overhead", in *IEEE Network*, page(s): 8 –16, July – Aug. 2000

54. S. Deering and R. Hinden, "Internet Protocol, version 6 (IPv6) specification",

RFC 2460, Dec. 1998