A Mobile Toolkit and Customised Location Server for the Creation of Cross-Referencing Location-Based Services

Submitted in partial fulfilment
of the requirements of the degree
Master of Science
of Rhodes University

Shange-Ishiwa Ndakunda

30 January 2013

Abstract

Although there are several Software Development kits and Application Programming Interfacess for client-side location-based services development, they mostly involve the creation of self-referencing location-based services. Self-referencing location-based services include services such as geocoding, reverse geocoding, route management and navigation which focus on satisfying the location-based requirements of a single mobile device. There is a lack of open-source Software Development Kits for the development of client-side location-based services that are cross-referencing. Cross-referencing location-based services are designed for the sharing of location information amongst different entities on a given network. This project was undertaken to assemble, through incremental prototyping, a client-side Java Micro Edition location-based services Software Development Kit and a Mobicents location server to aid mobile network operators and developers alike in the quick creation of the transport and privacy protection of cross-referencing location-based applications on Session Initiation Protocol bearer networks. The privacy of the location information is protected using geolocation policies. Developers do not need to have an understanding of Session Initiation Protocol event signaling specifications or of the XML Configuration Access Protocol to use the tools that we put together. The developed tools are later consolidated using two sample applications, the friend-finder and child-tracker services. Developer guidelines are also provided, to aid in using the provided tools.

Acknowledgements

I am truly indebted and thankful to my supervisors Mrs. Madeleine Wright and Prof. Alfredo Terzoli. Your time and effort dedicated in guiding me towards the completion of this thesis is priceless. This dissertation would not have been possible if I had not gotten the support from Dr. Mosioua Tsietsi, Ms. Mathe Maema, Mr. Zalalem Shibesi, Mr. Ray Musvibe and the entire Rhodes University Convergence team. I am truly grateful to many of my colleagues who supported me. Finally, thanks is due to the Rhodes University Computer Science Department and Centre of Excellence who provided me with all the resources required to carry out the research.

Contents

List of Tables

List of Figures

Listings

Glossary of Terms

API: Application Programming Interface

APPUSAGE: Application Usage

AUID: Application usage Identifier

BTS: Base Transceiver Station

CRUD: Create, Update and Delete

DHCP: Dynamic Host Configuration Protocol

GEOPRIV: Geographical Location/Privacy Working Group

GeoJSON: Geographical JavaScript Object Notation

GeoRSS: Geographical Rich Site Summary

GML: Geographical Markup Language

GPP: Generic Positioning Protocol

GPS: Geographical Positioning System

GLONASS: Global Navigation Satellite System

HELD: HTTP-Enabled Lcation Delivery

HTTP: Hypertext Transfer Protocol

IETF: Internet Engineering Task Force

IMS: IP Multimedia Subsystem

ILS: IMS Location Service

IP: Internet Protocol

iOS: iPhone Operating System

JAIN: Java APIs for Integrated Networks

JAXB: Java Architecture for XML Binding

JBOSS: JavaBeans Open Source Software Application Server

JME: Java Micro Edition

JPA: Java Persistence API

JSE: Java Standard Edition

J2ME: Java 2 Platform, Micro Edition

KML: Keyhole Markup Language

LAM: Location Aware Messaging

LAPOC: Location-aware Push-to-talk

LBSs: Location Based Service

LC: Location Client

LI: Local interface

LS: Location Server

LoST: Location-to-Service Translation Protocol

MD5: Message-Digest algorithm 5

MIME: Multipurpose Internet Mail Extensions

MIDP: Mobile Information Device Profile

MNOs: Mobile Network Operators

MPS: Mobicents Presence Server

NGN: Next Generation Networks

OMA: Open Mobile Alliance

OSI: Open Standards Interconnect

PDA: Personal Digital Assistant

PIDF: Presence Information Document Format

PIDF-LO: Presence Information Document Format - Location Object

PIR: Private Information Retrieval

POJO: Plain Java Object

PSAP: Public-Safety Answering Point

RA: Resource Adaptor

REST: Representational State Transfer

RFC: Request for Comments

RFID: Radio-Frequency Identification

RLS: Resource List Server

RPID: Rich Presence Information Data format

SBB: Service Building Block

SCE: Service Creation Environment

SDK: Service Development Environment

SDP: Service Development Platform

SIMPLE: Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions

SIP: Session Initiation Protocol

SLEE: Service Logic Execution Environment

SLO: Spatial Location

WAN: Wide Area Network

W-INFO: Watcher Information

UAC: User Agent Client

UAS: User Agent Server

URI: Universal Resource Identifier

XCAP: XML Configuration Access Protocol

XDMS: XML Documents Management Server

XML: Extensible Markup Language

XMPP: XML Instant Messaging and Presence Protocol

Chapter 1

Introduction

The rapid increase in the number of smart mobile communication devices as well as the growth of mobile broadband services have led to an increase in the data traffic that is used on mobile operator's networks [5]. This growing bandwidth consumption has resulted in consumer-driven markets where Mobile Network Operators (MNOss) no longer enjoy the benefits of the delivery of voice-centred services [7]. There is growing pressure for MNOss to aggressively and quickly provide rich and innovative consumer services. These new services may be a combination of multiple media and data such as voice, video, presence and location.

MNOss already have at their disposal Service Delivery Platforms and Service Creation Environments (SDPs and SCEs) to quickly and efficiently create, deploy and control personalised network-based multimedia services [7, 41]. One of the growing dimensions of multimedia services is location. Recent trends have indicated that both the user base and revenue of the location-based services (LBSs) market are expected to grow considerably in the near future as discussed in section 1.1.2. The rapid growth of LBSs can most likely be attributed to the increasing presence of affordable efficient hardware (such as GPS-enabled phones) and software to deliver these services cost-effectively and at acceptable privacy protection levels.

Eventhough LBSss are still not a significant generator of revenue for many MNOss, this project was designed to put together a client-side Java Micro Edition Location-Based Services Development Kit (JME LBSs SDK) and a Mobicents LBSs server to aid MNOss and developers alike in the quick creation of cross-referencing LBSs applications. The emphasis in cross-referencing LBSs is on the sharing of location information amongst different

LBSs entities on a given network. This chapter gives the main reasons for undertaking this research, the scope, the problem statement and the goals that we aimed to fulfill upon completion. The chapter concludes with a brief presentation of the layout of the rest of the thesis.

1.1 Research Motivation

1.1.1 The Importance of the Fast Development and Deployment of Services

To meet the growing demand for rich multimedia services, MNOss must form new strategic partnerships with key technology and development organisations [5, 7]. They should also expose network-side Application Programming Interfaces (APIs) for quick service creation and management. A good example of this is Verizon which started providing APIs for location and messaging services in September 2011 [81, 19].

In addition to the SCEs, MNOss may also need to provide client-side Service Development Kits (SDKs) for putting together client-based mobile device applications. These SDKs may aid developers in quickly developing and deploying client-based solutions to communicate with the services provided on the network in similar fashion to the Verizon NavBuilderInside (NBI) SDK discussed in [19].

1.1.2 Location-Based Services Growth and Development

Recently, trends have indicated that LBSs adoption and development are on the increase and will be even more widespread in the near future. According to Juniper Research, global market revenues for LBSs are expected to exceed $12 billion by 2014 [17]. The total user base of consumer LBSs is predicted to reach 1.4 billion by 2014, according to Gartner [50]. Gartner also named LBSs applications as one of the top ten consumer applications to watch for in 2012. On top of these predictions, smart phone LBSs applications and games such as Foursquare, Gowalla, Facebook and Google places, have all grown considerably in the past two years.

It is evident that MNOss could benefit tremendously from focusing more on the LBSs market. This opinion is based on the expected increase in the revenue and usage of LBSs

in the near future. The main motivation behind this research project is that MNOs need to have facilities to create and deliver LBSs applications quickly and efficiently. These facilities should be basic and reusable so that they may easily be integrated into multimedia applications.

1.2 Scope

Figure 1.1: Self-referencing location-based services, adapted from [26]

1.2.1 Self- and Cross-Referencing Location-Based Services

In the context of LBSs, client entities that generate location information are known as Targets while those that consume this information are known as Location Recipients or Clients (LCs) [66]. These two terms (i.e. Targets and LCs) are used extensively in this thesis to distinguish between these different LBSs client entities.

LBSs can be categorised as either self- or cross-referencing as discussed in [26]. The emphasis of self-referencing services is on satisfying the location information needs of single client entities mostly by querying network-based content and mapping providers. This is depicted in Figure 1.1. In self-referencing LBSs, the LC and Target entities are co-located at the same client terminal. An example of such an LBSs is an application that allows a user to find shops in close proximity.

Cross-referencing LBSs allow different client terminals (i.e Targets and LCs) to share location information with each other on a network. This is depicted in Figure 1.2. A Child Tracking service that allows parents to locate their absent children is a good example of a cross-referencing service.

Our scope is limited to addressing the communication needs of cross-referencing LBSs.

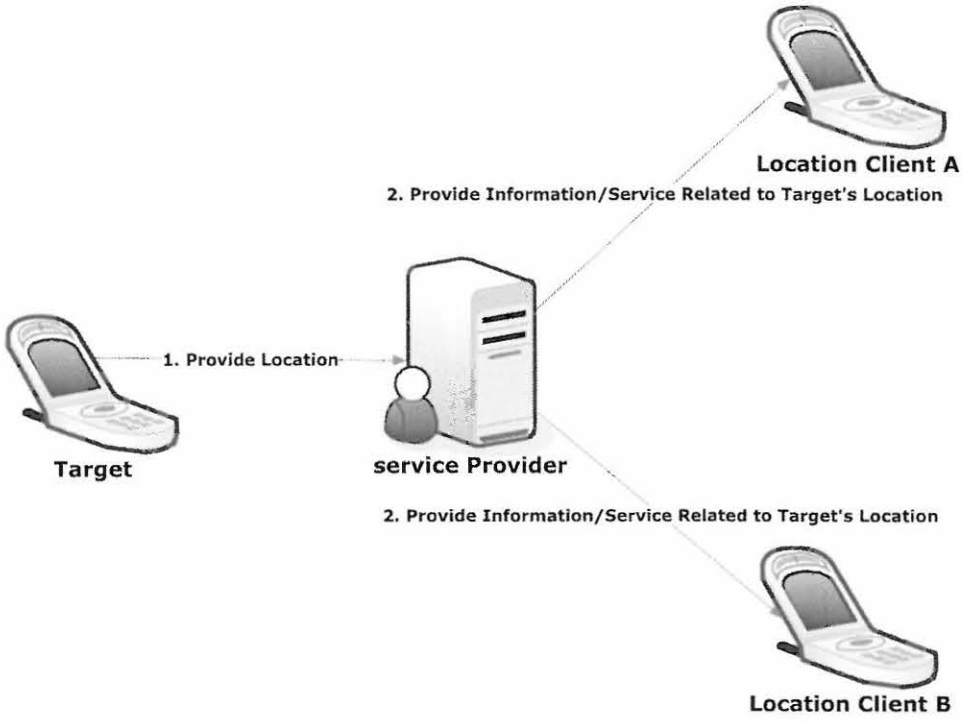

Figure 1.2: Cross-referencing LBSs, adapted from [26]

1.2.2 The Mobicents Service Creation Environment

The work presented in this thesis is part of a bigger research effort to set up a fully-functional Next Generation Networks(NGNs) testbed at Rhodes University. The testbed runs on the open-source Java-based Mobicents SCE by Redhat. For this reason, the work performed as part of our project was undertaken using Mobicents as the SCE.

Mobicents is designed around the idea of abstracting complexities of basic services such as presence and call-control by organising them into reusable Service Building Blocks (SBBs) or enablers [54]. For instance, to provide a quick and innovative service, a developer might use a presence SBB and a call-control SBB without getting caught up in the complexities of any of these SBBs. The new service's SBBs can then, in turn, be combined with other SBBs to build new services. This approach offers the opportunity to reduce drastically the time and effort needed during service creation and deployment. The aspects of Mobicents

that are important in understanding the contents of this thesis are discussed in Chapter 3.

1.2.3 Java Micro Edition

Work on this research project started at the beginning of 2009. At the time, around eighty percent of all mobile handsets shipped supported the mobile Java 2 Platform, Micro Edition (J2ME) [9]. It was the most popular development and run-time environment across the different mobile platforms available at the time. In addition to this, it provides good support for the composition of LBSs, Hypertext Transfer Protocol (HTTP) and Session Initiation Protocol (SIP) applications, through its libraries. We recognise that newer mobile platforms such as Android and iPhone Operating System (iOS) have gained a lot of popularity in recent years [18]. However, the iOS platform is vendor-specific and we had made a commitment to JME before Android got popular on many devices during 2010 and 2011.

1.3 Problem Statement

Mobicents provides SBBs for media signalling, registration, proxy, presence and call control. Mobicents does not provide any facilities for the development transmission and management of LBSs. A Mobicents location server composed of SBBs to manage the private communication of location information could easily be used to support the location needs of Targets and LCs and those of other multimedia services.

A second problem is that there are several SDKs and APIs for client-side LBSs development such as the Garmin LBSs toolkit; the Nutiteq Mapping SDK; Navteq APIs and SDKs (e.g. DeCarta); and the ArcWeb J2ME mobile toolkit. These toolkits are mainly focused on the creation of self-referencing services such as geocoding and reverse geocoding; creating and downloading map images; route management and navigation. Others like the Verizon NavBuilderInside SDK are MNOs-specific. So, there is a lack of open-source SDKs for the development of client-side LBSs that are cross-referencing.

1.4 Goals for the Thesis

This thesis seeks to address the problems mentioned in the preceding section through the attainment of the following goals:

1. Putting together a Mobicents location service (composed of SBBs) that can be used for the sharing of cross-referencing LBSs information amongst different client entities. This would allow Target entities to publish their location information and for LCs to subscribe to it. The SBBs that make up the location server should be reusable and extensible to support the quick creation of location-aware multimedia services.

2. Building a JME LBSs SDK for the quick development of client-side cross-referencing LBSs applications. The SDK will be used to develop Target and LC applications that share location information using the Mobicents location server.

3. LCs may be applications running on the mobile phones of close friends, family members or servers of marketing and government organisations. Therefore, Targets should be able to control which LCs see their location information and how it is shared amongst those LCs. The third goal involves putting together facilities to ensure the privacy of the sensitive information shared by the Targets.

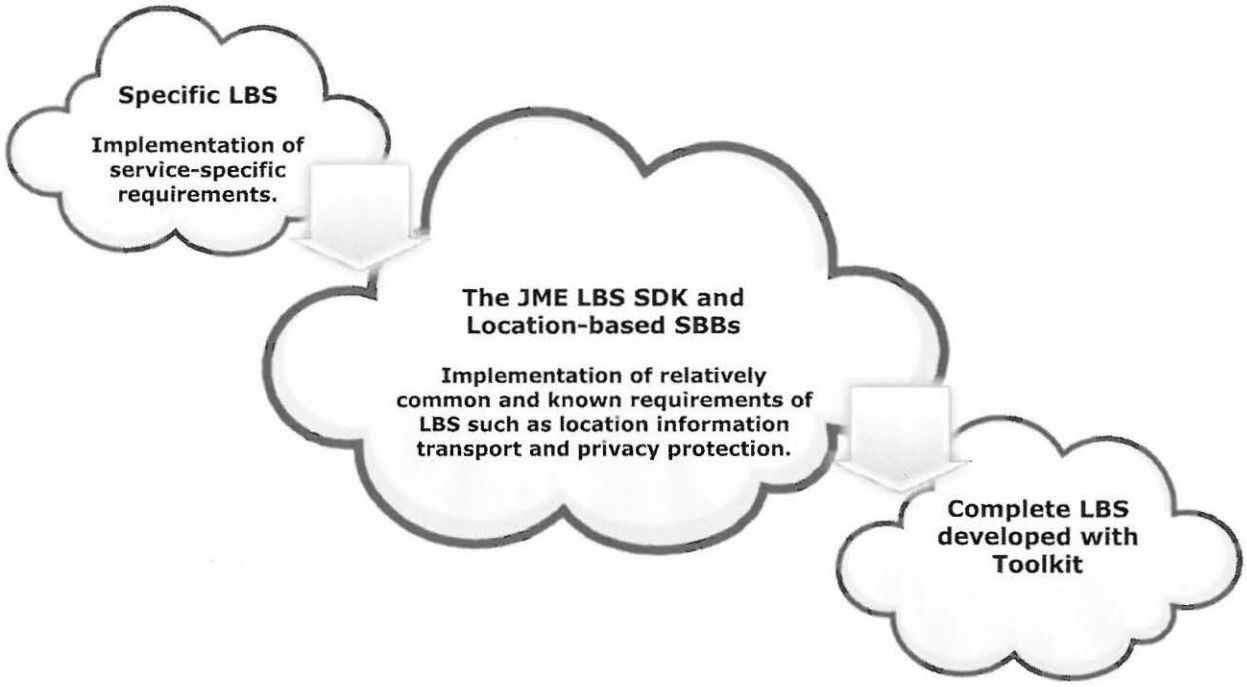

Figure 1.3: An overview of the envisaged location-based services toolkit

Figure 1.3 shows a picture of the goals of the envisaged client-side JME LBSs SDK and Mobicents SBBs for the quick creation of LBSs. Common LBSs transport and privacy requirements will be implemented as part of the client-side JME LBSs SDK and Mobicents SBBs. This will allow developers to focus on the core requirements of their cross-referencing LBSs, shortening development time in the process.

1.5 Methodology

The development approach used in this research project is incremental prototyping. The building blocks for the JME LBSs SDK and the Mobicents location service were designed and implemented incrementally. They were later integrated into a complete system.

Firstly, priority was given to developing the transport facilities of the JME LBSs SDK and on ensuring that these facilities work well with a Mobicents location server. Secondly, the privacy facilities on both Mobicents and the JME LBSs SDK were put together. Lastly,

the JME LBSs SDK was improved by providing interfaces and abstract classes to aid in the creation of different LBSs entities.

1.6 Thesis Layout

The remaining chapters of this thesis are organised as follows:

Chapter 2 introduces the different LBSs components and classification schemes and discusses previous work and concepts relevant to understanding the contents of this thesis. Approaches to LBSs transport and privacy are discussed. A commitment to providing server-side SBBs and a client-side SDK to support cross-referencing LBSs using SIP event signalling system is made. Finally, the usage of geolocation policies to ensure the privacy of the location information is defended.

Chapter 3 discusses aspects of the two environments, Mobicents and JME, that are needed to understand the work undertaken in our project. Firstly, important Mobicents SCE components such as SBBs are discussed. An introduction of JME components such as the location and SIP libraries then follows. The end of the chapter discusses the components that needed to be put together using the two environments to achieve the goals of the project.

Chapter 4 begins with a description of the JME LBSs SDK prototype and the modification of the Mobicents Presence Service (MPS) such that it supports location information signalling. The focus here is on enabling the transport of the location information between Targets, the modified MPS and LCs. The JME LBSs SDK prototype facilities that support the development of the location transport of LBSs are exposed to developers as part of one class. The chapter narrows the transport of the location information to only a few geodetic shapes and civic location levels.

The focus of Chapter 5 is on putting together JME LBSs SDK prototype components to ensure the privacy of the location information transport facilities discussed in chapter 4. The first part of the chapter discusses the addition of the geolocation policy application usage (appusage) to the Mobicents XML Document Management Server (XDMS). The geolocation policy appusage enables the geolocation policy documents of Targets to be stored at the XDMS. The second part focuses on the creation of JME LBSs SDK prototype facilities to help in creating, modifying and deleting geolocation policy documents at the XDMS. These facilities are exposed as part of one interface to the developers. The

last part of the chapter focuses on enabling the use of the geolocation policy documents to privately share the location information of Targets.

Chapter 6 consolidates the facilities of the JME LBSs SDK prototype put together in Chapters 4 and 5 by providing interfaces and abstract classes to aid developers in using them. Interfaces and abstract classes to help in the creation of Target, LC and policy-making entities using the JME LBSs SDK prototype are discussed here. The chapter also discusses the development of the friend-finder and child-minder services which were used both to add new components to the prototype and to showcase its functionality. The development of the prototype is finally frozen in this chapter.

Chapter 7 concludes the work done in this thesis by providing a brief discussion of the tools developed to meet the objectives stated in the introduction. The chapter also highlights the major contributions of the thesis. These contributions include client- and server-side tools that could be used by MNOs and developers alike to quickly create cross-referencing LBSs. Other tools aid developers in ensuring that the location information transmitted in LBSs is protected by using geolocation-policies. The chapter concludes by discussing the limitations of the tools developed and recommending future work that could be conducted to further improve the research.

Chapters 4, 5 and 6 focus on the building of the JME LBSs SDK and the modification of the MPS. Appendix A focuses on the services produced as part of the project and provides guidelines on how they could be set-up and used by developers.

Chapter 2

Related Work

In this chapter we provide a discussion of the literature that is pertinent to the private transmission of LBSs information on SIP networks. Firstly, we discuss the general components that make up an LBSs, looking at the important classification schemes used in categorising different LBSs. Secondly, we discuss the potential protocols that we have considered using to communicate LBSs information between the network-side Mobicents LBSs SBBs and the client applications that are developed using the JME LBSs SDK. We chose to use the SIP protocol for the transport of location information. Our discussion therefore focuses on the different architectures used in transmitting LBSs in SIP-based networks. Finally, we provide an overview of the different approaches to LBSs information privacy and focus particularly on geolocation policies.

2.1 Location-Based Services Components and Classification

2.1.1 Location-Based Services Components

LBSs can be realised through several architectures and produced and consumed through a plethora of devices. Most, however, are comprised of the same key components. According to [75, 12], these components are mobile devices, a communications network, positioning components and data and content providers. These are illustrated in Figure 2.1.

In surveying work related to ours we are going to refer to some of these components to

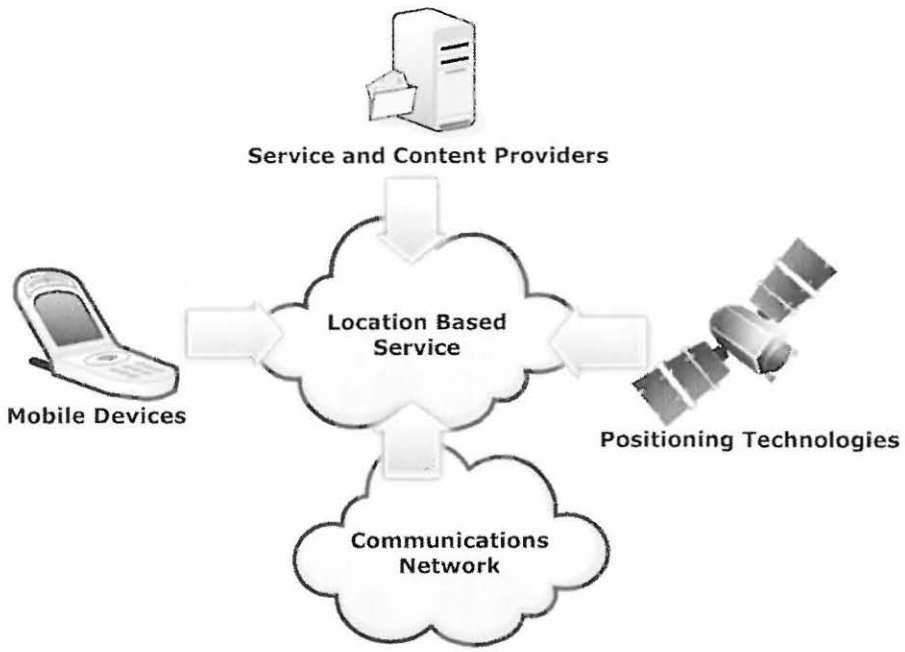

Figure 2.1: LBSs components, adapted from [12]

describe and analyse the different architectures related to our research. Looking at these components also provides us with a better insight into the organisation of our toolkit and SBBs.

2.1.1.1 Mobile Devices

The mobile devices in Figure 2.1 represent mobile phones, Personal Digital Assistants(PDAs), laptops and pads used by the end users to produce and consume LBSs. A mobile device may run software that may query location information from a service provider or other network entity. In such a case, the mobile device takes the role of an LC. Other software on a mobile device may also provide location information to be processed or used at a service provider or other network entity. In such a case the mobile device takes the role of a Target.

2.1.1.2 Positioning Technologies

The positioning technologies shown in Figure 2.1 refer to the hardware and software components that are used in determining the location of the Targets. These technologies include satellite positioning (e.g Global Positioning System (GPS), Global Navigation Satellite System (GLONASS)); cell tower positioning (such as that realised with the Ericsson Mobile Location API [11]); Radio Frequency Identification (RFID); and network-based protocols such as the Dynamic Host Configuration Protocol(DHCP). Location determination technologies are not the main focus of our work and we make an assumption that the location determination hardware and software is available at the Target. Our focus is on privately communicating the location information with other network entities.

2.1.1.3 Service and Content Providers

LBSs are provided by service providers that are capable of answering requests about the position of Targets. These services include navigation and routing to points of interest, emergency handling and yellow pages. Content providers own or lease content that can be used to develop more engaging applications [12]. Content distribution formats such as Geographical-Rich Site Summary (GeoRSS), Geographical-JavaScript Object Notation (GeoJSON) and Keyhole Markup Language (KML) are commonly used to communicate location-based information by content providers [12]. Our work focuses on allowing service providers and MNOss to create, deploy and manage LBSs easily. Content and mapping provision services are outside the scope of our work.

2.1.1.4 Communications Network

Last but not least, we come to the communications network which represents the hardware and software used for carrying the communications messages between the mobile devices and the service and content providers. The carrier network on which a mobile device resides plays an important role in the architecture used to communicate LBSs information to and from it. For our work, the focus is at the Open Systems Interconnection (OSI) model's application layer level. We discuss only issues that are relevant at this level.

2.1.2 A Taxonomy of Location-Based Services

The domain of LBSs is huge and we need a taxonomy for choosing the LBSs in which we are particularly interested. There are different classification schemes for LBSs and they may be used for different purposes. For example, [53] categorise LBSs along functional lines such as navigation, tracking and games. [64] discuss a scheme that classifies LBSs on whether they are device-oriented or person-oriented. For our purposes, however, we are interested in only four aspects of LBSs: push or pull; self- or cross-referencing; single- or multi-Target and centralised or distributed. The descriptions and discussions of these categories follow below.

2.1.2.1 Push and Pull Location-Based Services

Pull or reactive LBSs are based on the LC making the initial request for a service [85]. The service provider provides the LC with the requested information. An example of a pull-based LBSs is an emergency service that a user can invoke in an emergency situation.

Push or proactive services, on the other hand, involve a service provider *pushing* location-based information to a Target. This is usually performed upon an occurrence of a predefined location event. For example, if a user enters a particular area of town, an advertisement could be pushed to them indicating discounts offered by stores in close proximity as discussed in [63].

In this project, we are interested in facilitating the development of applications that support both pull- and push-based LBSs.

2.1.2.2 Self- and Cross-Referencing Location-Based Services

The distinction between self- and cross-referencing LBSs was provided in section 1.2.1.

2.1.2.3 Single- and Multi-Target Location-Based Services

[26] differentiate between single- and multi-Target LBSs. In single-Target LBSs the focus is on a single Target's position in relation to static features in which the user at the target might be interested. A child-tracking application serves well in exemplifying such an LBSs. The focus in such a service is on one Target, the child.

Multi-Target LBSs focus on more than one Target at a time and, most-likely, the relationships between those Targets as well. An example of a multi-Target LBSs is a fleet management service or a multi-user proximity service that allows users to see others who are close to them. In our project we are interested in developing tools to facilitate the development of both single- and multi-Target LBSs.

2.1.2.4 Centralised and Distributed Location-Based Services

The final classification scheme we are interested in differentiates between centralised and distributed (peer-to-peer) LBSs. Centralised LBSs are provided and managed by a centralised server. Peer-to-peer services, on the other hand, are self-organising, which means that location data is exchanged directly between LCs and Targets without the need of centralised servers. In our project, the interest is only in centralised LBSs since the Mobicents SCP is usually deployed on IP networks as a centralised signalling server.

In the next section, we discuss candidate protocols and LBSs communication architectures.

2.2 Candidate Protocols and Location-Based Services Communication Architectures

In the LBSs ' classification section, we indicated our interest in all categories of LBSs but self-referencing peer-to-peer ones. We indicated in the introduction that one of our environments of implementation is the Mobicents SCE. In this section, we focus on the candidate protocols and architectures that could be used for communicating cross-referencing LBSs information implemented using the Mobicents SCE.

[58] identifies the important dimensions of location-based information as:

- The time the location was determined;
- The identity of the Target;
- The location information itself;
- Privacy protection and the representation of uncertainty, if any exists.

The four application-level protocols that we considered are:

- The HTTP-Enabled Location Delivery (HELD);
- The Location-to-Service Translation Protocol (LoST);
- The XML Instant Messaging and Presence Protocol (XMPP);
- SIP with a Presence Information Document Format - Location Object (PIDF-LO) payload.

2.2.1 HTTP-Enabled Location Delivery

Standardised in the Internet Engineering Task Force Request For Comments (IETF RFC) 5985, the HELD protocol is used by the LCs without location determination technologies to query their location information from a Location Server (LS). The LS on an access network such as a LAN or WLAN contains the location of the network devices and computers.

In this project we make the simple assumption that a mobile device has the relevant location determination technology. Therefore, we do not use HELD as a communications protocol for our work. However, we recognise that HELD could play an important role in querying the location information of other Targets on the network [39]. HELD requests can be used in novel ways for cross-referencing LBSs that allow LCs to get location information of the Targets they are interested in. This approach, however, is out of the scope of this dissertation and may be considered as future work.

2.2.2 The Location-to-Service Translation Protocol

There are scenarios where an LC, given its current location, may need the location and contact information of particular service providers on the network. For instance, a user might be in an emergency and need to make a call the nearest Public Service Answering Point (PSAP) as described in [3]. A PSAP is a public service office such as a police or fire station or an ambulance service. Even in non-emergency situations, the need might arise to call the closest product retailers [82]. The LoST is the IETF protocol used to perform these tasks [68, 16].

LoST is particularly useful in emergency services provision for PSAP discovery. Other uses of LoST, such as discovering the closest retailers are self-referencing. As we mentioned earlier, self-referencing LBSs are outside the scope of our work. It therefore was not appropriate to continue considering this protocol.

2.2.3 The Extensible Markup Language Instant Messaging and Presence Protocol

Another protocol we looked closely at was the XMPP. XMPP is standardised by the XMPP Standards Foundation and the IETF for purposes of presence, instant messaging, and real-time communication across the Internet [24].

The whole architecture is based around the publish-subscribe mechanism. The presence entity (presentity) broadcasts their presence information. Presence watchers subscribe to the published information. The XMPP Extension Protocol-0080 (XEP-0080) is a specification tailored specifically to location transmission [24]. The schema specified there transmits all of the three dimensions of LBSs communications we mentioned earlier: time, identity and location. The SIP Communicator discussed in [73], implements a geolocation aware contact list that conforms to this Specification.

An examination of the XEP-0080 Specification exposes some shortcomings regarding the time, identity and location that could be communicated:

- For the time dimension: the timestamp allows only for the expression of the time when the location of the Target was determined. It does not allow for the expression of the period during which the location information is valid or for how long recipients are allowed to retain it.

- For the location information: the Specification allows only for the expression of a pair of coordinates as the location of a Target (when expressed spatially). The accuracy field in the specified schema expresses the horizontal inaccuracy in metres. There is no provision for the transmission of other two dimensional and three dimensional shapes. The human-readable location information (known in the LBSs context as civic location information) has levels ranging from country to room level. A single location can be expressed as a combination of civic and geodetic (spatial) formats.

- In terms of privacy, XEP-0080 proposes that a Target should not publish its actual location. Instead, the published location information masks the actual location by introducing deliberate errors. The deliberate error for a given Target should be the same for all LCs to minimise the likelihood of triangulation by potential attackers. In our opinion, this might be a limitation in implementing some LBSs as the Target might want to share the actual location information with some of the LCs. We will discuss different privacy approaches in a later section.

2.2.4 Transmitting Location in Session Initiation Protocol-based Systems

The IETF and its child body known as the Geolocation and Privacy Group (GEOPRIV) have put forward standards for location information transmission in SIP-based systems. The XML document used to encode presence information in such systems is known as the Presence Information Document Format (PIDF) [14]. To include location, XML elements from namespaces specified by the GEOPRIV are added to the PIDF document. The elements are collectively known as a Location Object. The abbreviation of the document then changes to PIDF-LO.

2.2.4.1 Advantages of Session Initiation Protocol-based Systems Over Extensible Markup Language Instant Messaging and Presence Protocol from a Location Point of View

The PIDF-LO document provides fields to carry the three dimensions of location information namely time, identity and location. By studying the schemas and comparing them with those of XMPP location extensions, the following points may be noted:

- For the time dimension: the PIDF-LO allows for not only the time stamp of when the location information was determined, but also a means of expressing the time the recipients are allowed to retain it, or have access to it. It is therefore more expressive than the XMPP alternative.
- For location information: the Location Object of the PIDF-LO allows for more resolution than the XEP-0080 Specification for both the geodetic and civic location options. The geodetic option uses a special subset of Geographical Markup Language (GML) that allows for the communication of not only simple coordinates, but also other two-dimensional and three-dimensional shapes. Uncertainty can be expressed with the GML Geoshape circle.
- The PIDF-LO document schema has special fields for privacy related information. These allow for the indication of how long an LC is allowed to retain the location information. Another value specifies whether the recipients are allowed to retransmit it to other network entities or not. In addition, an IETF draft focuses on the private

distribution of a PIDF-LO document amongst different LCs in terms of the Target's geolocation policy. This will be discussed later in the privacy section 2.3.3.1.

It is evident that, from a location point of view, SIP-based standards specify a more expressive and private format than those provided for XMPP.

2.2.5 Choice Of Protocol

From the candidate protocols we have discussed, we have given reasons why we did not select the HELD and LoST protocols. That leaves us with two candidate protocols for the transmission of LBSs information over the Internet, XMPP and SIP.

From the comparisons in the previous section, we have concluded that XMPP location extensions are both less expressive and less private then those specified for use in SIP-based systems. However, one can still argue that XMPP might have superior real-time signalling capabilities and we can modify XMPP location extensions to be as expressive and as private as those specified for SIP-based systems. In section 2.2.5.1, we provide a comparison between XMPP and SIMPLE from the instant messaging and real-time signalling perspectives. SIMPLE is the SIP-based alternative to XMPP for instant messaging, real-time signalling and presence.

2.2.5.1 A Comparison of Extensible Markup Language Instant Messaging and Presence Protocol and Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions

In this section we begin the comparison between XMPP and SIMPLE by highlighting the main advantages that XMPP as a real-time and messaging protocol has over SIMPLE (which is SIP based). The advantages are as given below:

- XMPP is mostly XML-based and it is more compact and lightweight than SIMPLE, and as a result it is more scalable. SIMPLE is based on SIP. Therefore most information is conveyed in the headers and a larger number of specifications (RFCs) need to be implemented for SIP stacks, XML document management and instant messaging systems. This may be disadvantageous especially on mobile systems which are low on resources [83].

- XMPP is more widely adopted on the Internet than SIMPLE. One of the reasons for this is that XMPP was the initial instant messaging and presence protocol. SIMPLE was realised later to bring XMPP-like functionality to SIP-based systems [83]. XMPP is more popular amongst web-based companies like Google while SIMPLE is expected to gain more popularity amongst MNOss and telecommunication vendors [27].

Next, we highlight the advantages that SIMPLE has over XMPP with regards to our objectives. These advantages are:

- SIMPLE is based on SIP. It is inherently closer and more easily integrated into a telecommunications platform such as Mobicents than XMPP. SIMPLE applications may also inherit other SIP aspects such as authorisation, authentication and compression (SigComp) [27, 83].
- SIMPLE has been selected as the main instant messaging protocol for the IP Multimedia Subsystem (IMS) and it is predicted that many MNOss are going to opt for it over XMPP [83, 27].

The advantages of SIP-based systems in the area of transmitting location and ensuring privacy are more pertinent to our objectives than the compact lightweight environment provided by XMPP. Additionally, SIP-based location extensions offer more opportunities for integration into telecommunications applications. We chose to implement our LBSs components using SIP-based systems instead of XMPP-based systems.

2.2.6 A Discussion of Location-Based Information Transport Over Session Initiation Protocol Networks

Having chosen SIP as the main protocol for the purposes of transmitting location, we present the standards, architectures and previous work undertaken in this area. All the approaches discussed in this section transmit the location information as part of the PIDF-LO document mentioned previously. We provide a brief discussion of this document first. Thereafter, we discuss the different approaches to location transmission. These approaches are grouped into three general categories: SIP location conveyance; location information transmission reusing the presence event system or package; and location transmission using a presence-independent location event system or package.

2.2.6.1 The Presence Information Document Format with a Location Object

Presence provides "*real-time information regarding the willingness and ability of a presence entity (presentity) to participate in communications*" [3]. The secure transport and communication of location data over the Internet is essentially the same as that of presence [49]. Therefore presence documents (PIDF) are reused for the carriage of Location Objects (PIDF-LO). As explained earlier, the Location Object carries the various location aspects that a Target shares in civic or geodetic format [79].

The PIDF-LO document is a better and more expressive alternative as compared to the Spatial Location (SLO) format, earlier defined by the IETF and used in implementations in [6]. SLO does not use GML and offers little resolution for expressing locations. [44] defines a custom Generic Positioning Protocol (GPP) for location transport. GPP supports GML for different positioning mechanisms but it has not been standardised, and PIDF-LO offers richer extensions.

Next, we take a look at the different approaches to location information sharing in SIP networks.

2.2.6.2 Simple Location Conveyance in Session Initiation Protocol Systems

The approaches taken by [38, 6, 52] propose the carrying PIDF-LO documents in SIP messages (i.e INVITE, OPTIONS, REFER, UPDATE, MESSAGE, SUBSCRIBE, PUBLISH, REGISTER, BYE). A SIP request should contain a *Geolocation* header indicating that it carries location information or a reference to it. The general scenario is depicted in Figure 2.2. A SIP INVITE request is conveyed with the location information (PIDF-LO), through intermediary servers to the LC. 200 OK SIP messages are sent in response as an acknowledgement of their arrival.

[6] also proposes location conveyance by means of SIP REGISTER requests. The approach used embeds SLO documents in the body of SIP requests and helps Targets discover their location from the LS. [69] also proposes the use of SIP REGISTER requests to indicate a device's presence in a logical entity such as a conference room.

From our observations, the disadvantages of this approach are as listed below:

- The Target initiates communication to the LC through an intermediary server. This might be a limitation for some cross-referencing LBSs such as friend finders where the

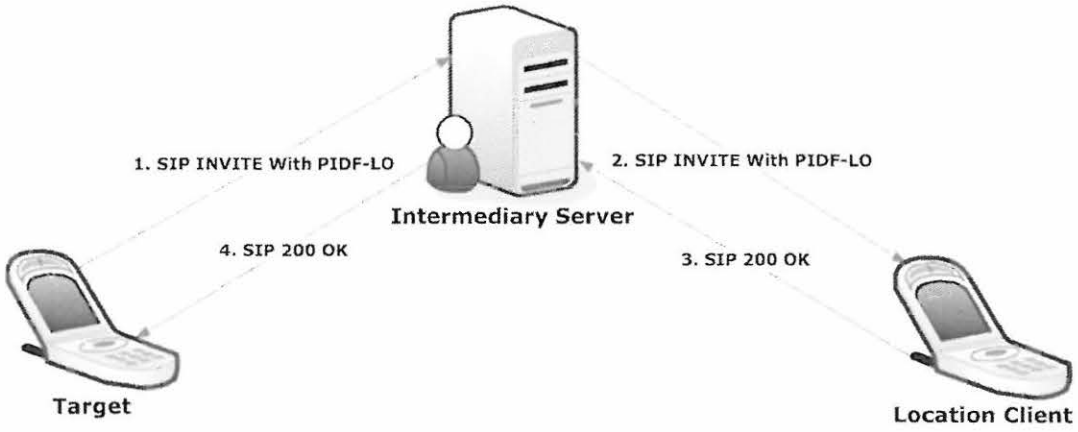

Figure 2.2: Location conveyance using SIP INVITE requests, adapted from [52]

LC is required to pro-actively acquire the location of the Target. The SIP location conveyance approach provides no mechanism for LCs to initiate dialogues to Target to get their location information.

- There is a general absence about the support of location-sharing privacy in these architectures apart from the routing privacy specified in [52].

Because of the aforementioned reasons, we stopped considering this approach to transmit LBSs information for our system. In the next section, we discuss an approach of transmitting LBSs information in SIP presence systems which are based on a SIP event signalling system.

2.2.6.3 Location-Based Session Initiation Protocol Presence Systems

Some LBSs implementations in SIP-based systems treat location as a special extension to presence information. This means that they communicate the availability and willingness to communicate of a Target or presentity. [21, 22, 69, 72, 2] are some of the LBSs implementations based on the presence event package (SIP presence system) in SDPs such as IMS. [48] propose the use of peer-to-peer presence agents that are enhanced with location. [2] implement a friend-locator prototype using a location-enhanced presence service. The Ericsson IMS-Innovation project in [10] also takes a similar approach, for an Android IMS client.

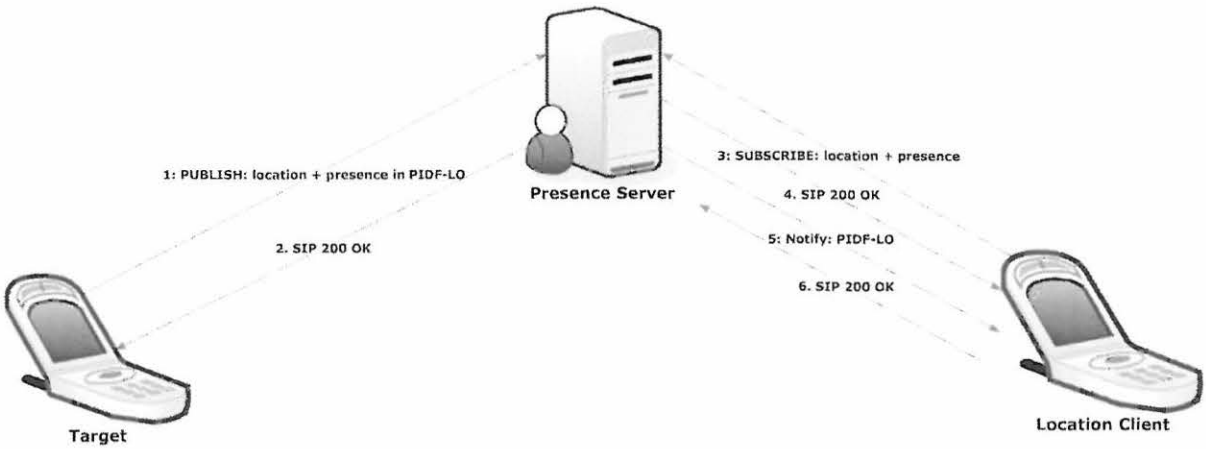

Figure 2.3: Location transmission using the presence event package, adapted from [45]

The general flow of messages in these implementations is as depicted in Figure 2.3. The Target (which is also a presentity in presence terms) publishes its location together with its presence information in a PIDF-LO document at step 1. The LC (which is also a presence watcher) subscribes to that information in step 3. The LC is then notified every time a new publication is made by the Target to the Presence Service. For notifications from multiple Targets, the LC may subscribe to resource lists (such as a buddy list) instead of subscribing to individual Targets as discussed in [21].

Conveying location and presence in the same messages may be advantageous as explained in the following points:

- LCs and the service providers readily have both types of data (presence and location) and do not need to maintain states of two separate event systems (packages). The combination of the two data types can lead to the development of more informative context-aware applications. This is well exemplified in the vehicle tracking solution developed at Columbia University [72] and in the call-control example implemented in [86].

- Although no literature was found that focuses on the bandwidth utility benefits derived from the combination of location and presence in one event system, it is apparent that combining presence and location might utilise less bandwidth than if separate event systems were used. If presence and location updates and notifications are performed at the same time in the same SIP messages for the LC and Target, com-

paratively less bandwidth may be used. This may be the case with sending different updates and notifications from different location and presence event signalling systems as discussed in [39]. This assumption is based on the fact that the SIP headers on the different messages will be associated with their own overhead, and so using one SIP message instead of two will be more bandwidth-thrifty.

Combining presence and location in a single event package also has disadvantages, as listed in the following points:

- Some applications may have different requirements for location and presence notifications. By design, presence is specified by picking values from a pre-defined list of alternatives. Location, on the other hand, can be taken from a range of contiguous values as spatial or civic locations. For location we may have to specify what location changes are enough to warrant notifications [38]. For presence information, notifications may be performed after a certain presence element has changed to another. Conflicts may arise, especially in toolkit and service creation environments where a wide range of requirements need to be catered for.
- SBBs and enablers are supposed to be basic so that they can be easily and efficiently reused in putting together other services. If presence and location are in one event package, then a service that requires only presence cannot have it without the overhead of the location system. This may be disadvantageous.

2.2.6.4 Presence-Independent Session Initiation Protocol Location-Based Services

The Open Mobile Alliance (OMA), instead of reusing the presence event package, specifies a whole new event package exclusively for location-based event signalling [37, 45]. Location events are carried within the PIDF-LO document, but no presence information is included as explained in the previous section. The general flow of events is as described in section 2.2.6.3. However, publications and subscriptions are made to a Location Server as depicted in Figure 2.4. [37] discusses the work-in-progress of the implementation of this approach in the IMS.

[38] also implement a presence-independent enabler, namely the IMS Location Service (ILS) in the IMS Service Layer. Any service that wants to request location within the

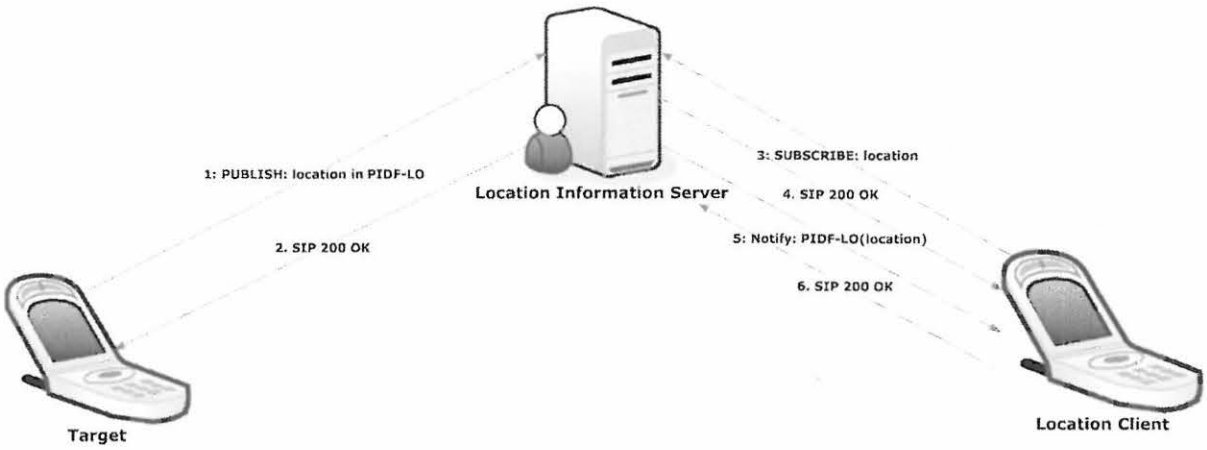

Figure 2.4: Location transmission using the location event package, adapted from [45]

IMS Service Layer can use SIP SUBSCRIBE requests to the ILS. The ILS notifies the subscribing applications with the information they have requested. This ILS enabler provides an opportunity for new value-added LBSs to be composed within the IMS. The prototype services used to test and realise the ILS are the Push-based Location Aware Messaging (LAM) and the Location-aware Push-to-talk (LaPoC) Services.

2.2.6.5 Choosing a Location Transmission Approach and Architecture

We have discussed in section 2.2.6.2 that the SIP location conveyance is mostly suitable for self-referencing LBSs. We have also given other reasons why we have not chosen that approach. That leaves us with either reusing the SIP presence system or using a separate SIP location event signalling system. At this point of the project, we refrained from making a choice between these two approaches. Since Mobicents provides a presence service (MPS), we made a decision to implement a location system that reuses this MPS rather than putting together a presence-independent location system from scratch. In future, a separate Mobicents location service can easily be extracted from our end product.

Before we move on to location privacy, we discuss location event notification filtering. This is a mechanism that may enhance SIP location event systems to provide only the relevant information. Location event notification filtering is critical in enabling push-based LBSs.

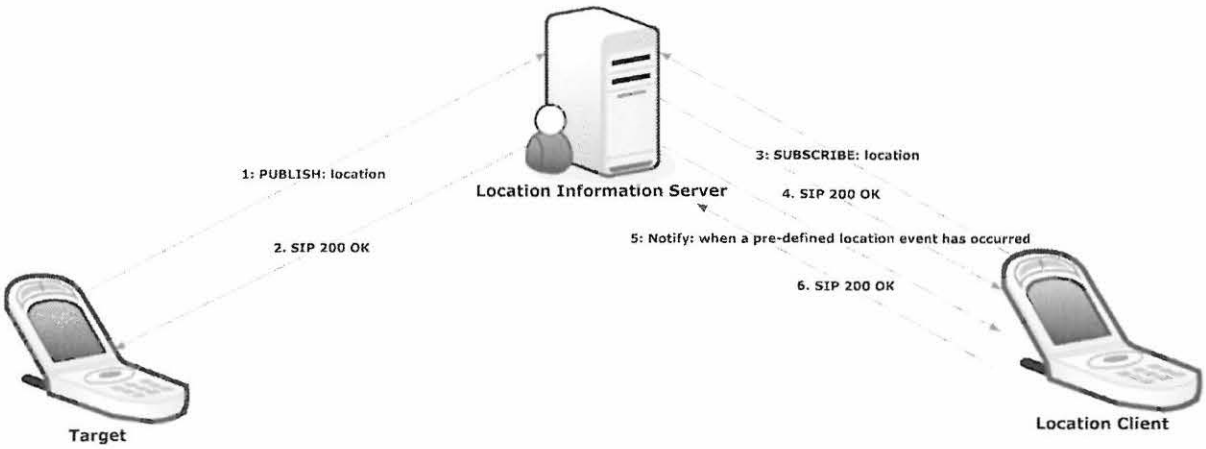

Figure 2.5: The flow of events for location event notification filtering. adapted from [78]

2.2.6.6 Location Event Notification Filtering

In SIP event systems, notifications are made to the subscribers when the information subscribed to changes. Notifications are also made upon subscription and subsequent subscription refresh requests (discussed in Chapter 4). According to [38, 78] not all location-based systems can rely on this mechanism as different applications might require different location resolutions and may have critical times for notification. [48, 38, 63, 25] propose a filtering solution that reduces the transmission of updates over the air in Cellular networks. This is done by specifying events that should warrant notifications.

In SIP event systems, location event notification filtering provides the means for LCs to indicate the kind of location events they are particularly interested in. The general location filtering scenario is depicted in Figure 2.5. Location filters are conveyed as XML documents within SIP SUBSCRIBE request payloads. Notifications are sent to an LC only when the events, pre-defined in the filters, have occurred.

[26] point out that the querying or polling is suitable for pull cross-referencing LBSs and periodic event-based (pre-defined events) updates for push LBSs. We have also made the observation that in many SIP and IMS implementations, location filters are key to implementing cross-referencing push-based LBSs. [38] uses filters for the push-based LAM and the LaPoC Service. [2] also uses filters for a location-based messaging application known as LocalNote and [63] uses them for pushing information to Targets in location-based marketing application.

To conclude the review of work related to ours, we discuss the different approaches regarding LBSs privacy.

2.3 Location Privacy

Location is a very sensitive form of information and [29, 87] highlight the importance of privacy in any form of LBSs. Approaches such as pseudononymisation and anonymisation attempt to hide the identity of the user to protect their privacy during location queries. Privacy policies provide the means to control the information shared between Targets and LCs. In this section, we take a brief look at these different approaches and their relevance to our objectives.

2.3.1 Pseudononymisation

Pseudononymisation is an approach to location privacy that attempts to mask the identities of the LCs or Targets by using false identity information (pseudonyms) [40]. [70] discuss architectures that have a trusted service provider or third party which provides pseudononymisation to the mobile device. This trusted third party may become the single point of failure. To counter this shortcoming, [58] employ an architecture where a trusted service provider is given the time, location and pseudonym but not the actual identity of the mobile device. The pseudonym is changed periodically at the mobile device. The mobile device then communicates the pseudonym and, at times, its identity to third-parties who are authorised to see it. The mobile device has to make alternative connections to these third parties to inform them of the pseudonym they are using at a given time.

[4] recognise the shortcoming of pseudononymisation: it concentrates only on the identity part of the location information. This could be problematic especially in anonymous self-referencing LBSs because the location and time of the LC are exposed. To counter this false locations are introduced. The mobile application sends its correct location to the server in a set of other false locations. The server responds to every one of the locations in the set and the client picks the correct response.

The reasons why we do not use pseudononymisation are as follows:

- By design, this solution looks mostly suitable for self-referencing pull-based LBSs which are out of the scope of our project.

- For our purposes the overhead of privacy provided by pseudononymisation may be high for both the server and the mobile applications, especially in SIP systems where identities may be used for purposes other than location.

- Although pseudononymisation is a sound approach to location privacy, we decided to focus more on how a Target could vary its location information amongst different LCs. This is a practical concern in real life social networks as specified in [1].

2.3.2 Anonymisation

Another approach to location privacy is anonymisation. This approach provides privacy by ensuring location queries cannot be linked to the LCs from which they originated. A trusted location anonymiser, which may become the single point of failure as in pseudononymisation, is located between the LC and the service provider and aggregates user requests to be processed by the service provider at once. The anonymiser processes the request of a particular LC by grouping it with *k-1* real or dummy requests, removing any identity information; changing location data using cloaking algorithms and only then sending these queries to the service provider. This approach makes the LC's location indistinguishable at least amongst other *k-1* users at the LBSs provider [53, 70].

Anonymisation has weaknesses. Careful analysis of context information about the k requests may give away the information about the users. Also attacks can be launched by introducing malicious requests into the k requests for a particular area [70] . In fact, [70] reported that even with anonymised location data it could be possible to predict locations up to ninety-three per cent of the time. [15] points out the weaknesses of anonymisation and proposes Private Information Retrieval (PIR) as the solution. The approach utilises cryptographic techniques that do not require third parties that may present a single point of failure. As previously mentioned, cryptographic techniques are out of the scope of our project. Anonymisation is a sound approach towards providing location privacy. However, we do not use it in our project because of the following points:

- From our observations, anonymisation is more suitable for self-referencing pull-based services because it is all about LCs querying for locations.

- Like pseudononymisation, this approach also does not focus on how a Target could vary its location information amongst different LCs.

In the next section, we discuss an approach that uses privacy policies which is more in line with our project objectives.

2.3.3 Privacy Policies

Users are more concerned about the privacy of cross-referencing and tracking LBSs than they are of self-referencing ones [29]. Research by Google has shown that in social networks people disclose information according to levels of trust [1]. In location information disclosure, the important factors from the end-user's point of view are: their current location, their current general context, the identity of the requester and the time of the request [66]. Privacy policies are defined to allow users to specify their location-sharing preferences in line with all or some of these factors.

[53] propose a richer privacy policy architecture with more information regarding the context of the Target and LC. This includes factors such as the role of the requester and the reason for the location request (e.g commercial, administrative). Another viewpoint of [53] is that anonymisation, pseudononymisation and privacy policies could be combined for more private location-sharing solutions. [8] propose a similar architecture that approaches privacy by varying the location precision of an LC based on the trustworthiness of different service providers. In our project we provide privacy by using geolocation policies which are tailored to work with the SIP event signalling system.

2.3.3.1 Geolocation Policy

The IETF and the OMA specify the geolocation policy as an extension of common policy defined in [66, 45, 67]. It is designed to allow Targets to specify their preferences pertaining to the location information they publish in the PIDF-LO documents described earlier [79]. Geolocation policy parameters are specified within an XML document. We shall discuss more about the document structure in chapter 5.

We have chosen to use geolocation policies to provide location privacy because:

- They are more suitable to cross-referencing LBSs as the Target explicitly specifies the LCs that are allowed to see its information.
- They focus on how a Target may vary its location information amongst different LCs.

- The approach is designed to work with the SIP event signalling system, so it can be implemented for presence and location event packages discussed earlier.

The question that logically follows from this discussion is: how does the Target specify the privacy policies (geolocation policy)? The answer is: by using the XML Configuration Access Protocol (XCAP).

2.3.3.2 The Extensible Markup Language Configuration Access Protocol

Each Target may own a privacy policy document that is stored at the XDMS. Storing these documents at the XDMS allows the presence or location services to access them during location information subscription as implemented in [21]. This also allows users to use multiple clients as the privacy policy documents could be stored on the XDMS and can be downloaded as long as the client is authenticated. The communications protocol used by the Targets to perform the creation, updating and deletion (CRUD) operations on their documents, located at the XDMS, is known as the XCAP [23, 21].

XCAP is based on HTTP and works by constructing Uniform Resource Identifiers (URIs) identifying whole XML documents, their elements or their attributes for CRUD purposes. The Targets take on the role of XCAP clients to communicate CRUD operations to the XDMS. The XDMS may have an interface for serving XCAP requests known as the XCAP Server.

2.3.3.3 Application Usages

It is not only geolocation privacy policy documents that may be stored on the XDMS. Other data for other applications such as presence privacy policy and buddy lists can be stored there as well. In the XCAP context, the specific applications for which the documents stored on the XDM server are being managed are known as application usages (appusages). Following from this definition, geolocation policy and presence policy are both appusages. Buddy lists fall under the resource-lists appusage [61, 21].

2.4 Chapter Summary

We have reviewed the literature that is pertinent to the private transmission of location-based information in SIP core networks. At the beginning of the chapter, we provided

a discussion regarding the classification of LBSs and put emphasis on the categories in which we are interested, namely centralised cross-referencing LBSs. We then reviewed the available standards, technologies and protocols and made decisions regarding their importance in meeting our goals. We made a decision to start using the SIP event signalling system that combines presence with location because Mobicents already provides a presence service. In future, a separate location system may be extracted from such a system. Finally, we discussed the different approaches to LBSs privacy. The approach we chose is that of privacy policies because they are more relevant to cross-referencing LBSs than other approaches.

Chapter 3

Software Environments Used

This chapter provides a discussion of the important aspects of our implementation environments, namely the Mobicents SCE and JME. We use Mobicents as the host and development environment for the server-side LBSs application. The central part of the Mobicents SCE is the JAIN-SLEE Core, so we briefly discuss it first. We then look at other important components such as Resource Adapters (RAs) and SBBs. We emphasised in Chapter 2 that we decided to start our server-side implementation by modifying the MPS and provide privacy protection using the XDMS. We provide a discussion on the MPS and XDMS components and the SBBs that they comprise. For the client-side environment, the JME, we focus on SIP, Location and HTTP APIs that are available for the development of the JME LBSs SDK. Finally, we highlight the modifications that should be performed on the MPS and the components that should be put together on JME to meet the goals of our project.

3.1 The Mobicents Java APIs for Integrated Networks - Service Logic and Execution Environment

Mobicents is a Java-based SCE for NGN applications [54]. It is based on the Java APIs for Integrated Networks - Service Logic and Execution Environment (JAIN-SLEE) [13]. JAIN-SLEE is a Java component model and architecture that is designed to provide high throughput, low-latency and fail-over capabilities to event-based applications [56]. Through JAIN-SLEE components called RAs, Mobicents can work with data from various IP proto-

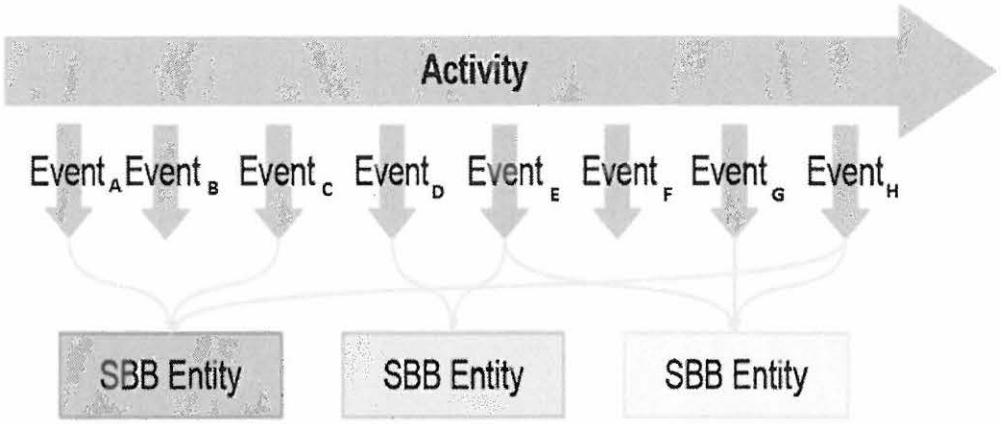

Figure 3.1: Events, activities and SBB entities [28]

cols. At the time of writing (in June 2011) , Mobicents had SIP, HTTP, Diameter, SMPP and XMPP RAs. For the creation and composition of services, Mobicents uses JAIN-SLEE components known as SBBs. In the next subsection, we provide a brief discussion of the JAIN-SLEE Core, the Mobicents event routing engine.

3.1.1 Event Flow in the Java APIs for Integrated Networks - Service Logic and Execution Environment

The JAIN-SLEE Core runs on top of JavaBeans Open Source Software Application Server (JBoss) middleware and it is the central part of the Mobicents SCE. It is an application server comprised of components that are optimised for event-driven applications. When events are received by Mobicents from the network, they are processed by the RAs. The RAs adapt the data in particular protocols, such as HTTP into Java objects known as SLEE Events. RAs then fire these SLEE Events to the SLEE for handling [13].

Every SLEE Event has an Event Type that determines the routing of the instances of that Event Type in SLEE. The SLEE Event Router controls the flow of these asynchronous SLEE Events and passes them to SBB Entities that are interested in them. SBB Entities are logical instances of specific SBBs (discussed in 3.1.3). A related stream of events is known as an Activity [13, 28]. The forwarding of SLEE Events in an Activity to interested SBB Entities is illustrated in Figure 3.1.

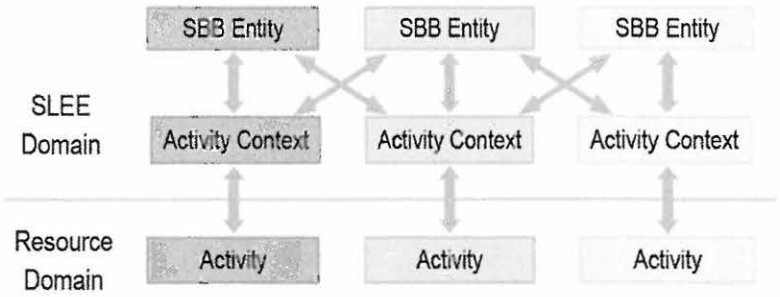

Figure 3.2: SBB entity and resource adapter communication [28]

3.1.2 The Resource Adapters

[13] defines resources as systems that are external to the SLEE. This may include protocol stacks such as SIP, HTTP, Diameter, XMPP, network devices that are in communication with SLEE, and databases. As mentioned before, RAs serve the function of facilitating communication between the SLEE and these external resources. RAs define Activities on which related Events could be fired and received [28]. As shown in Figure 3.2, the SLEE encapsulates these Activities in Java objects known as Activity Contexts.

For instance, when a SIP request arrives at the SIP RA, the data it carries is converted to a SLEE Event that is fired on the SIP Activity Context for processing. Also, when SLEE wants to communicate back to the Resource after the Event processing is finished, it invokes the services of the SIP RA on an Activity Context to convert the SLEE Event to a SIP Response or Request.

3.1.3 Service Building Blocks

SBBs contain the logic that is used to process incoming SLEE Events and generate responses that are fired back to the SLEE. Initial requests may also originate from the SBB to the SLEE. As we mentioned earlier, an instance of a particular SBB is known as an SBB Entity. SBBs can use Event Type Identifiers to indicate interest in receiving and firing SLEE Events of that type. SBBs define event-handler methods to process the incoming SLEE Events; local Interfaces to facilitate asynchronous communications with other SBBs; and SBB life-cycle methods [13].

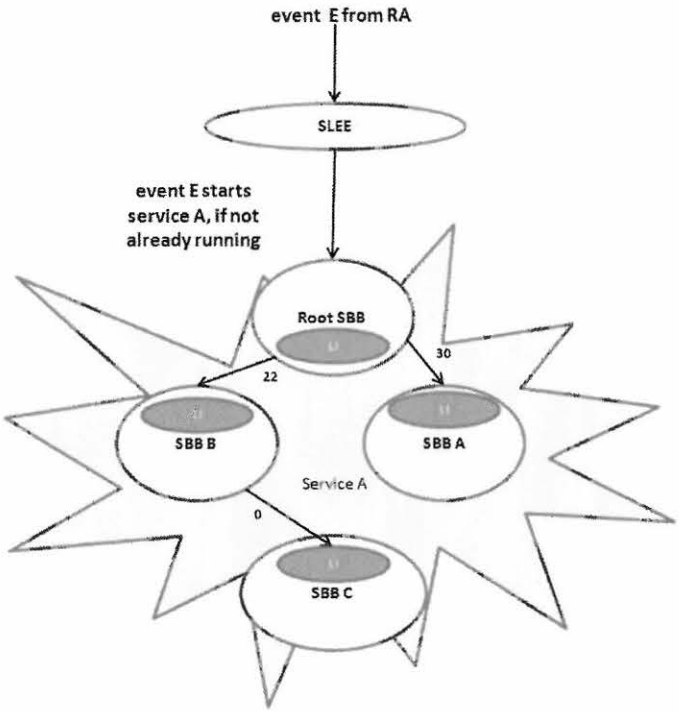

Figure 3.3: SLEE services and event forwarding , adapted from [13]

SBB Entities are organised in logical tree structures. They may communicate with each other using SLEE Events forwarded according to pre-defined priorities. SBB Entities may also form relationships to other SBB Entities and invoke their local Interface methods asynchronously [13].

3.1.3.1 Service Creation and Event Forwarding in the Service Logic and Execution Environment

To create an SLEE Service, the developer must create SBBs and organise them into a logical tree, based on the order in which SLEE Events of interest will be forwarded to them (using priority values). They must then indicate one or more SLEE Event that may cause SBB Entities to be instantiated by the SLEE (events that start the service). Figure 3.3 shows a hypothetical SLEE service, A. The SLEE receives an event, E, that is forwarded to service A. When event E is received by the root SBB of service A, the service is started if it is not already running.

The root SBB of service A processes the event and first forwards it to SBB A for further processing. The event is forwarded to A first because it has the highest priority (30) amongst the children of the root SBB. After SBB A finishes processing the event, it is forwarded to SBB B and its subtree. Each SBB has a Local interface (LI) which is SBB logic (code) that facilitates the extension and reuse of SBB code in other SBBs [13]. Any SBB can synchronously invoke the LI of another SBB in the same tree. Service developers can leverage this facility to create and compose new services using existing SBBs and is one of the main reasons why we chose Mobicents as our SCE.

3.1.4 The Mobicents Presence Service and XML Document Management Server

Mobicents provides the Mobicents MPS for SIP presence event signalling and service creation [55]. The Mobicents XDMS provides services such as the management of privacy protection rules to the MPS. Since we opted to use the MPS as the starting point for our server-side location services, it is important that the reader has a good understanding of how the MPS and XDMS works.

3.1.4.1 The Mobicents Presence Service and XML Document Management Server Components and Their Interactions

Figure 3.4 shows the MPS and the XDMS. The MPS is responsible for managing the publications, subscriptions and notifications to the SIP presence clients (i.e Presence Source and Watcher). SIP presence clients may be applications that are either internal of external to the MPS. Client applications developed with the envisaged JME LBSs SDK will be classified as SIP presence clients as well. The XDMS manages authorisation policies (e.g. presence rules) [62].

Figure 3.4 also shows the interactions between the different components. The Presence Source communicates its authorisation rules (for privacy protection) to the XDMS by using the XCAP. The communication between the SIP presence clients and the MPS is through the SIP protocol. Presence Sources publish their presence information to the MPS and Presence Watchers subscribe to the MPS to get this information through notifications. Internal server communication involves the MPS acquiring authorisation rules from the XDMS so they can be used to control subcriptions and notificaions to the Presence Watcher.

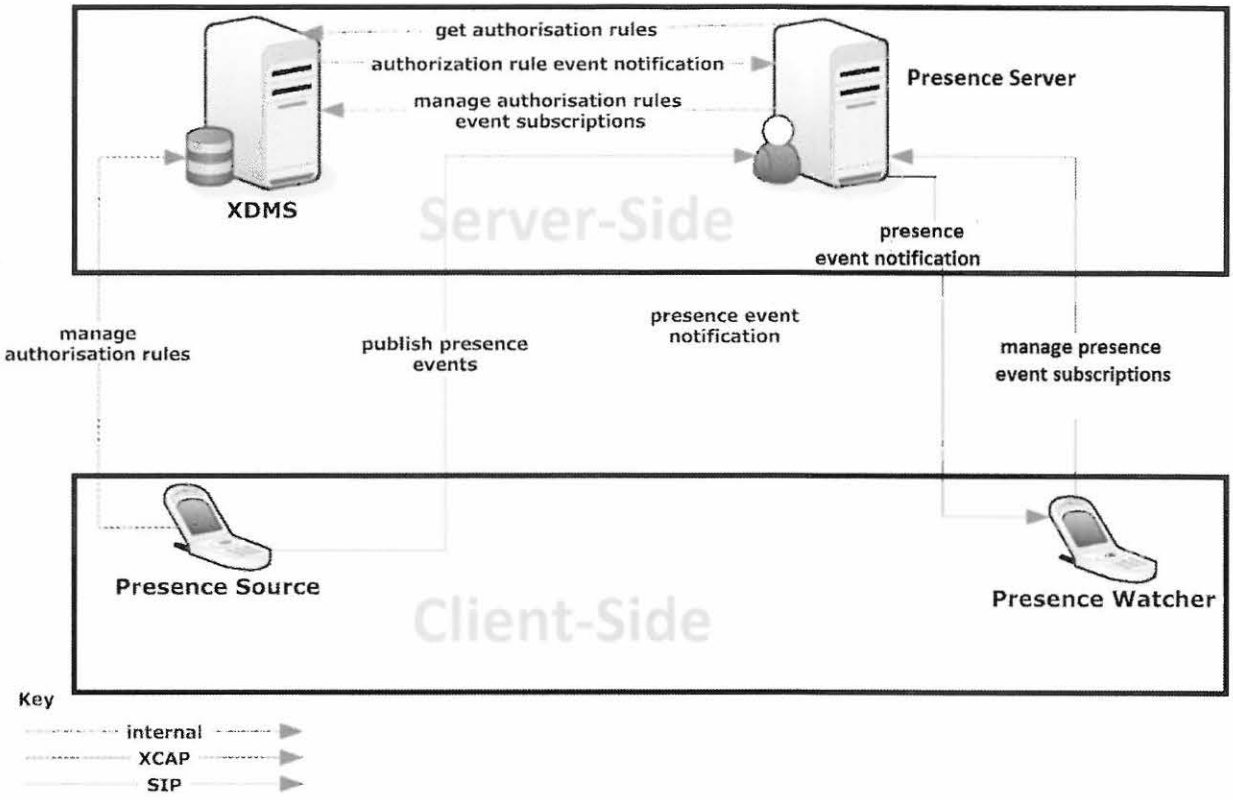

Figure 3.4: A simplified MPS and XDMS and client-side components, adapted from [55]

3.1.4.2 The Mobicents Presence Service and XML Document Management Server Service Building Blocks Overview

The MPS components shown in Figure 3.4 and the relationships between them are implemented as SLEE Services. The equivalent SBB trees are as shown in Figure 3.5. For simplicity, we show only the main SBBs and we also completely leave out those SBBs that belong to the XDMS. For the MPS, the root SBBs are the `PublicationControlSBB` and the `SubscriptionControlSBB`.

The SLEE passes SLEE Events generated from the SIP PUBLISH requests to the `PublicationControlSBB`. Events generated from SIP SUBSCRIBE requests, on the other hand, are provided to the `SubscriptionControlSBB`. The `SubscriptionControlSBB` generates SLEE notification events that it sends to the SLEE. The SLEE then sends these events to the SIP RA so that a SIP NOTIFY request can be generated and sent.

The PublicationControlSBB, SubscriptionControlSBB and the EventListSubscriptionControlSBB (which is a part of the RLS) are implementations that are built around the generic SIP event package framework specified in [57]. Therefore, they handle generic SIP functionality such as maintaining publication, subscription and notification dialogs.

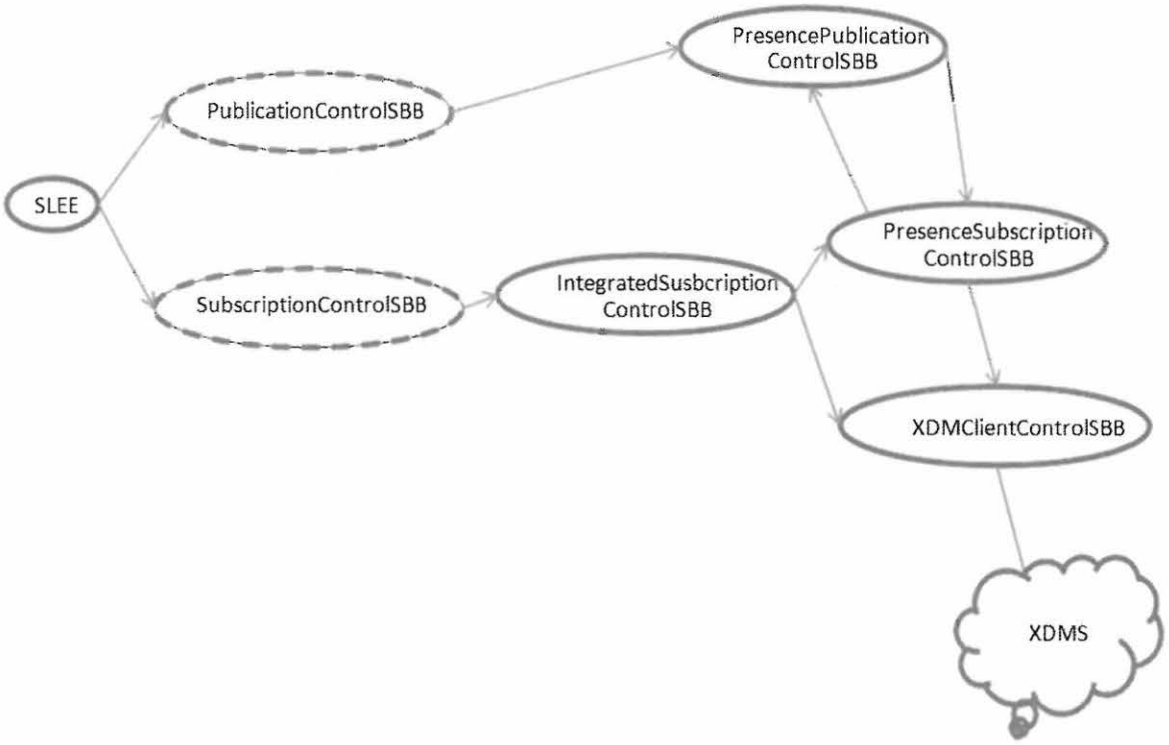

Figure 3.5: An overview of the MPS SBBs

The PresencePublicationControlSBB is invoked asynchronously by the PublicationControlSBB (using its local interfaces) to process presence specific functions. These functions include authorising publications as well as parsing and validating PIDF documents that carry presence. The PresencePublicationControlSBB has a relation to the PresenceSubscriptionControlSBB to notify it with the published presence information (obtained from the PIDF documents). New PIDF documents are composed at the PresenceSubscriptionControlSBB from the published presence information to be sent to different subscribers. Information pertaining to all current publications is maintained in a database using a Hibernate Java Persistence API (JPA) implementation.

On the subscription side, methods of the IntegratedSubscriptionControlSBB are in-

voked asynchronously by the `SubscriptionControlSBB` upon occurrence of SIP subscription events. The `IntegratedSusbcriptionControlSBB` then invokes methods of the `PresenceSubscriptionControlSBB` to perform presence specific functions. These functions includes authorising subscriptions and others we will describe in the next chapter. The `XDMClientControlSBB` allows the `PresenceSubscriptionControlSBB` to access authorisation rules (presence policies) from the XDMS. Information pertaining to all current subscription dialogs and authorisation rules are also maintained in a database using a Hibernate JPA implementation.

We have discussed the JAIN-SLEE components and basic operations. We have also discussed the MPS up to the level of SBB and their functions. We use the MPS as the starting point for our envisaged LBSs SBB. In the next section, we discuss the JME, the environment in which we developed the client side of the toolkit for building LBSs applications.

3.2 The Java Micro-Edition

The Java Standard Edition (JSE) provides a lot of functionality and features that are not suitable for limited mobile and embedded systems. JME provides a development and run-time environment for Java software targeted at these limited systems [84]. JME was formerly J2ME. For the purposes of our research, we are interested in the libraries of JME that will enable us to perform central cross-referencing LBSs information communication. These are the Location API, the SIP API and the HTTP client.

3.2.1 The Location API

"*In a quest to popularise and enable LBSs on the JME Platform, JSR179 defines a set of generic APIs that can ease the development of LBSs applications* " [33]. JME provides LBSs abstraction for developers in the `javax.microedition.location` package. The `LocationProvider` class abstracts the problem of location determination. Location could be determined through technologies such as satellite positioning or Base Transmitter Station (BTS) identification. The developers specify the location accuracy, response time, altitude, and the speed that they require from the `LocationProvider`. The `LocationProvider` class then invokes the appropriate available location determination technology from the underlying hardware and provides a one-time response or periodical notifications.

The API also provides a mechanism for the storage of landmarks that can be very useful for self-referencing LBSs.

3.2.2 The Session Initiation Protocol API

With expectations of the SIP standard to become popular on mobiles devices, the JSR180 specification defines a SIP stack and API for SIP application development in JME. These components are built compactly enough to allow them to run on devices with small memory and processing capabilities. The interfaces and classes are contained within the `javax.microedition.sip` package [32].

As shown in the Figure 3.6, a device (or application) may contain client and server capabilities as it is customary in SIP implementations. The client part may be known as the User Agent Client (UAC) and the server part as the User Agent Server (UAS). UACs may use SipClientConnection instances to connect to UASs that may use instances of the `SipServerConnection` interface. Instances of the `SipConnectionNotifier` listen to incoming connections from other UACs [32].

The demo applications in this area include an implementation that allows two users to invite each other to SIP sessions and communicate through JME sockets [51]. At the time of writing, no JME application of SIP presence or location sharing amongst Targets and LCs was found. There is a specification of the SIMPLE protocol for JME, but there is no implementation for it yet.

3.2.3 The HTTP Client

JME also provides an HTTP client API that allows HTTP connections to be established easily. The XCAP protocol that is used for the creation, modification and deletion of documents on the XDM server is based on HTTP [23, 21]. At the time of writing, there was no available implementation of an XCAP client for JME. We needed to perform significant modifications and building to put together a reusable XCAP client.

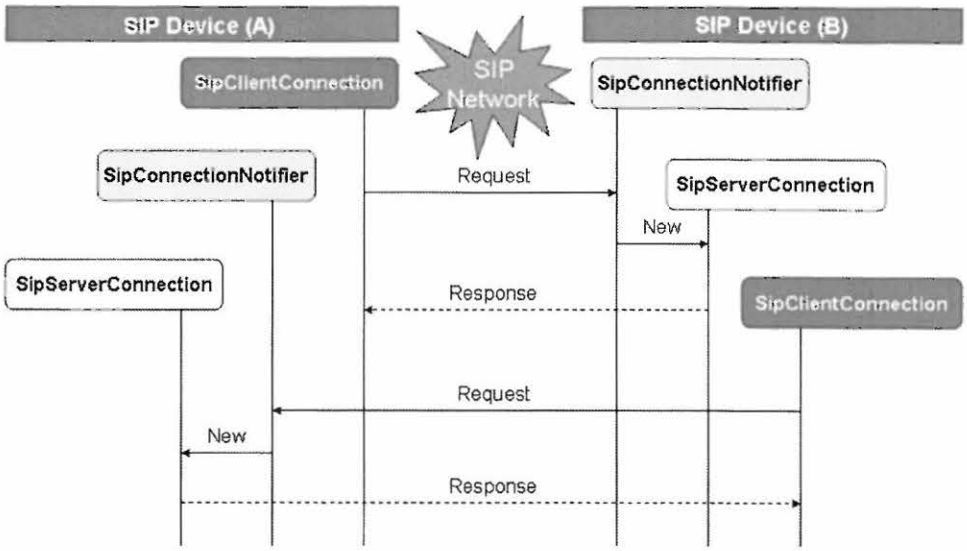

Figure 3.6: The JME SIP library [32]

3.3 Components That Needed to Be Created

3.3.1 Mobicents

In this subsection, we discuss the components that needed to be created to enable central cross-referencing LBSs in the above-mentioned environments. These are the components that we needed to put together to meet our objectives.

3.3.1.1 Location Communication and Transport

The MPS gives most of the functionality for presence services. There is currently no support for LBSs on the MPS. We needed to add or modify MPS SBBs so that they sufficiently communicate and manage cross-referencing LBSs information. Target components need to be able to publish their location information using PIDF-LO documents to the MPS and LCs require the ability to subscribe to this documents.

3.3.1.2 Location Notification Filtering

There was no support for location notification event filtering on the MPS. As we explained in Chapter 2, location event notification filtering is a crucial component for enabling push

LBSs. We therefore needed to add this functionality to the MPS.

3.3.1.3 Providing Location Privacy

The MPS supported presence privacy policies to ensure that the information belonging to presence entities is shared privately amongst watchers. The implementation was still not complete, however, as it did not apply the provided policies to the data (PIDF documents) published by the presentities. To meet our objectives, we first needed to add support for the geolocation-policy appusage (introduced in Chapter 2). This would allow Targets or other rule-making entities to create, modify and delete their geolocation-policy documents at the XDMS. Secondly, we needed to ensure that the geolocation-policies ar applied to the published location data (PIDF-LO) during LBSs subscriptions and that it had an effect on LBSs subscriptions.

3.3.2 The Java Micro Edition

3.3.2.1 Location Communication and Transport

In JME, there was no support for SIP presence services. There were also no open-source implementation of JME SIP presence User Agents available for reuse on the Internet at the time of writing. We had to put together UA facilities that are capable of SIP event signaling to use them to perform Target and LC functionality. The facilities put together needed to be organised into an SDK that can be used to compose centralised cross-referencing LBSs entities with ease. Applications developed with the SDK must work with the location services provided on the MPS.

3.3.2.2 Geolocation-Policy Support

As we mentioned earlier, JME provides an API for the easy implementation of HTTP client connections. On top of this, we needed to build facilities for XCAP client functionality so that Targets and other rule-making entities could create, modify and delete geolocation-policy documents on the XDMS. The facilities put together for XCAP client functionality should be part of the JME client SDK. Naturally, applications developed with the SDK must work with the geolocation-policy appusage that will be added to the Mobicents XDMS.

3.4 Chapter Summary

In this chapter, we have given a description of our implementation environments. First, we discussed the Mobicents SCE, because this is the environment that we will use to host and develop our server-side location services. We briefly introduced the JAIN-SLEE Core which is the central part of Mobicents. We then focused on several other components such as RAs and SBBs that are crucial in SLEE event processing. Thereafter, we provided a discussion of the MPS SBBs that are important in understanding our work. Our client-side LBSs SDK is to be developed using JME. We discussed the Location, SIP and the HTTP client APIs in JME that will be used in the private communication of central cross-referencing LBSs. Finally, we discussed the components that needed to be put together in these environments to meet the goals of our research project.

Chapter 4

Supporting Presence and Location

Chapters 2 and 3 emphasised our decision to start implementation by using the already-existing MPS. The MPS implements the SIP presence event signaling system (the SIP presence event package) as we have discussed it in Chapter 3. The initial step towards meeting our objectives was to design and implement a JME LBSs SDK prototype that works with the MPS to communicate basic presence information. Thereafter, we introduced location data (location object) handling. Note that the term *presentity* is used in this chapter to refer to a network entity that generates and publishes presence information. The term *watcher* is used to refer to entities that subscribe to this information [14]. In this chapter, the facilities put together on the JME LBSs SDK prototype are organised in such a way that they are accessible to an LBSs developer using an instance of a single Java class or interface.

4.1 The Session Initiation Protocol Registration Process

The JME client applications need to make SIP registrations to a SIP registrar service that binds SIP identities to accessible IP addresses [59]. Mobicents already provides a SIP registration service as discussed in [35]. Therefore there was no need to modify anything on the network infrastructure to enable these registrations. The process of composing and managing SIP registrations on JME is very similar to that of managing subscriptions and publications. Therefore, we discuss this SIP registration process in detail and explain the

aspects that are different for publications and subscriptions in later sections (4.2 and 4.3).

4.1.1 Session Initiation Protocol Registration Message Flow and Requirements

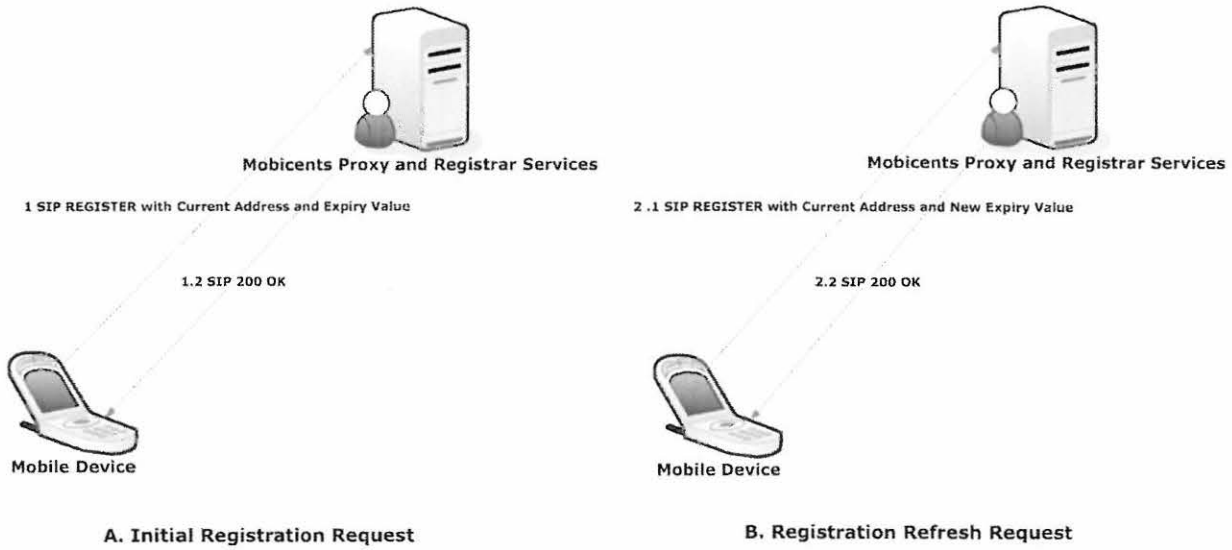

Figure 4.1: The SIP registration process, adapted from [59]

Before we look at the design and implementation of the `RegistrationClient` class, which we created in JME, we provide the general SIP message sequence we are trying to support as specified in [59, 35]. The JME `RegistrationClient` class composes a SIP REGISTER request with the headers displayed in Table 4.1. Step 1.1 in part *A* of Figure 4.1 shows such a request being sent to the Mobicents Proxy and Registrar Services. After such a request, a binding between the SIP address and the contact address and port of the mobile device are created at the Registrar Service. The *Expires* header holds the value expressing the time, in seconds, that should pass before this binding is considered invalid, and the registration dialog subsequently terminated.

The SIP request refresh process is required to keep the bindings valid and up to date to ensure that the mobile device is still alive at the client end. The process (of refreshing requests) goes on for as long as the client application at the mobile device is on-line. Step

Header	Description
To	The URI of the SIP Proxy to which registrations are being made.
From	The URI of the registering UA.
Contact	The current contact address and port of the UA
Expires	The time for which the contact information for the URI is valid before refreshing.

Table 4.1: SIP headers and values for registration requests, adapted from [59]

2.1 in part *B* of Figure 4.1 depicts the SIP registration refresh process. When de-registering, the value of the *Expires* header is set to a value of zero.

4.1.2 Implementation on the Java Micro Edition

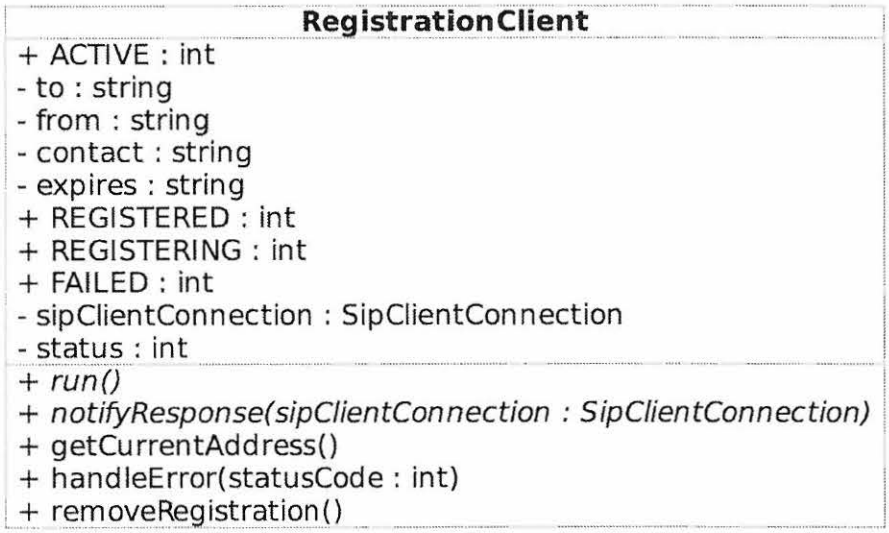

Figure 4.2: The `RegistrationClient` class

4.1.2.1 The Initial Registration Request

In this section, we discuss the implementation of the functionality required to perform the initial registration request (step 1.1 in Figure 4.1). First, we created the `RegistrationClient` class on JME containing the variables with *getter* and *setter* methods representing the headers shown in Table 4.1. The class is depicted in Figure 4.2. In its constructor, it uses an instance of the JME `SipClientConnection` class, discussed in section 3.2.2, to set the URI and the SIP headers for the initial request shown in Listing 4.1.

The process is performed as explained at the JME SIP library documentation found in [42].

The `contact` variable holds the current contact address of the device which is acquired using the `getCurrentAddress` method. The initial request is sent to the Mobicents Proxy and Location Registrar using the `send` method of the `SipClientConnection` instance. We set the `status` variable of the `RegistrationClient` class to the state of `REGISTERING` as an indication that an instance of the class is still registering and no response has been received yet.

```
1  try {
2                 scc = (SipClientConnection) Connector.open("sip:" +
                    proxyAddress);
3                 scc.initRequest("REGISTER", null);
4                 scc.setListener(this);
5                 scc.setHeader("From", from);
6                 scc.setHeader("To", to);
7                 scc.setHeader("Contact", contact);
8                 scc.setHeader("User-Agent", "LBSsToolkit");
9                 scc.setHeader("Expires", expires);
10                scc.send();
11     setStatus(REGISTERING);
12 } catch (Exception ex) {
13                ex.printStackTrace();
14 }
```

Listing 4.1: Code extract from the `RegistrationClient` class constructor

4.1.2.2 Receiving Responses

This section provides a discussion of the implementation of the functionality required to receive registration responses (such as the SIP 200 OK shown at step 1.2 in Figure 4.1). The `RegistrationClient` class implements the `SipClientConnectionListener` interface (discussed in Chapter 3) to handle SIP responses. In the code extract shown in Listing 4.1, the line `scc.setListener(this);` (line 4) indicates that the `RegistrationClient` class is the `SipClientConnectionListener` for SIP responses to requests made by its `SipClientConnection` instance, `scc`. The `SipClientConnectionListener` interface requires

the implementation of the `notifyresponse` method.

The code extract in Listing 4.2 shows how the `notifyResponse` method is implemented. At line 5, the `scc receive` method receives the incoming responses. The SIP registration response code is checked at line 9 for success. If it is successful, we set the status of the `RegistrationClient` to *REGISTERED*. The value of the `expires` variable of the class is set to the expiry value proposed by the Mobicents Proxy and Registrar Service in the response. Otherwise, if the registration request was not successful, we set the status to `FAILED` and invoke the `handleError` method that handles general SIP response errors.

```
1  public void notifyResponse(SipClientConnection scc)
2  {
3   try
4  {
5   scc.receive(0);
6   int responseCode = scc.getStatusCode();
7
8    System.out.println("[Registrar]: Registration Response: " + scc.
       getStatusCode());
9    if (responseCode == 200)
10   {
11    setStatus(REGISTERED);
12    if (!scc.getHeader("Expires").equals(null))
13    {
14      setExpires(scc.getHeader("Expires"));
15    }
16    } else {
17      setStatus(FAILED);
18      handleError(responseCode);
19    }
20 ...
```

Listing 4.2: The `notifyResponse` method code extract from the `RegistrationClient` class

4.1.2.3 Refreshing Requests

The `RegistrationClient` class must periodically refresh the registration information (bindings) of the mobile application at the Mobicents Proxy and Registrar Services (shown at step 2.1 in Figure 4.1). The `RegistrationClient` class extends the `Thread` class and implements the `SipRefreshListener` interface discussed in [43]. Listing 4.3 shows that a `SipRefreshHelper` instance, `refreshHelper`, is used to refresh the requests in the `run` method of the `RegistrationClient` class (`Thread`). The `refreshHelper update` method takes in parameters that should be updated from the previous SIP requests. For registrations, these parameters are the current expiry value and the current contact information (for updating the bindings at the servers).

The `getCurrentContact` method of the `RegistrationClient` class returns a `null` result if the contact information (SIP address and port) of the mobile device has not changed. Otherwise, the contents (new SIP address and port) returned by the `getCurrentContact` method are added to the refresh request. For ending the registration dialog with the server (going offline), the *Expires* header is set to a value of zero in the `removeRegistration` method (shown in Figure 4.2).

The `RegistrationClient` thread sleeps for a period equivalent to the value of the `expires` variable of the `RegistrationClient` class and runs for as long as the registrations are still successful. Using the `sleep` method of the Thread is practical only for a single refresh task such as SIP registrations. In section 4.4.2.2, we discuss how we synchronize the different refresh tasks (registrations, publications and subscriptions). If the value of the `expires` variable is zero, then the the `scc` connection to the Mobicents Proxy and Registrar Services is closed using the `closeConnections` method of the `RegistrationClient` class.

In the next section we move on to supporting SIP presence publications on JME.

```
1 public void run() {
2  while (status == REGISTERED || status == REGISTERING) {
3   try {
4    if(getCurrentContact()==null)
5      refreshHelper.update(refreshID, null,null
6      , 0, Integer.parseInt(getExpires()));
7    else
8     refreshHelper.update(refreshID,getCurrentContact(),null
9     , 0, Integer.parseInt(getExpires()));
```

```
10      if(Integer.parseInt(getExpires())== 0)
11        closeConnections();
12      Thread.currentThread().sleep(Integer.parseInt(getExpires())*1000)
          ;
13  } catch (Exception e) {
14  e.printStackTrace();
15  }
16  }
```

Listing 4.3: The run method code extract from the `RegistrationClient` class

4.2 The Session Initiation Protocol Publication Process

4.2.1 Session Initiation Protocol Publication Message Flow and Requirements

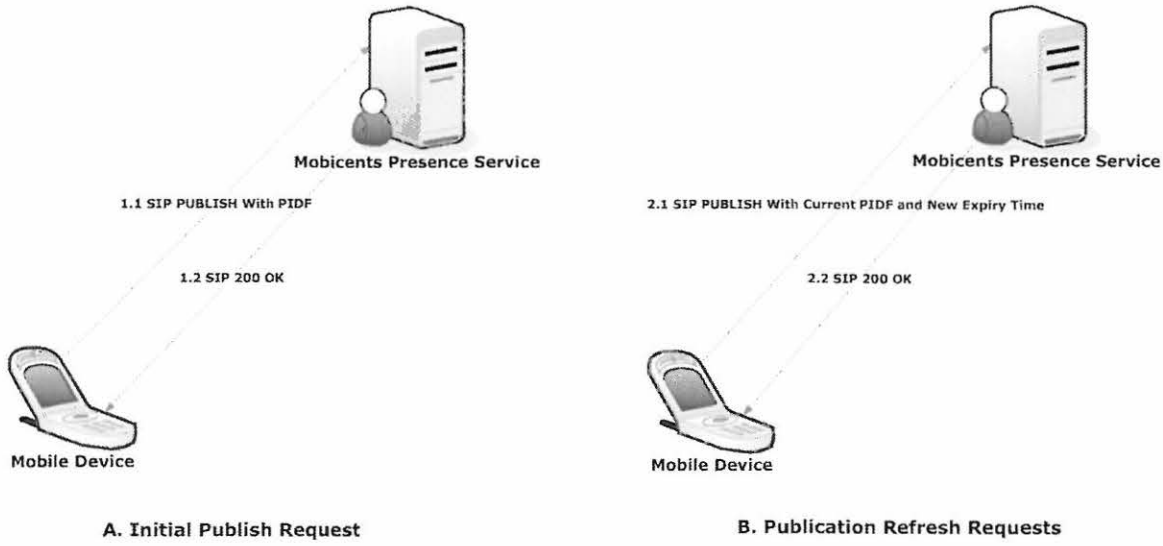

Figure 4.3: The SIP publication process adapted from [14]

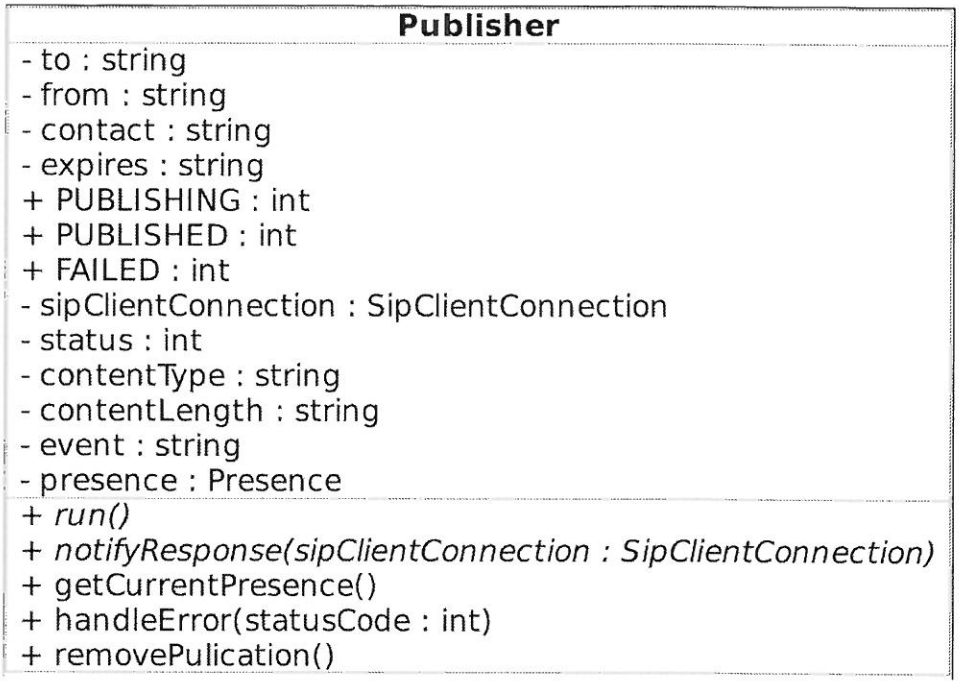

Figure 4.4: The `Publisher` class

After implementing the registration process, we proceeded with implementing the SIP publication process. PUBLISH requests contain *Content-Type* and *Content-Length* headers to describe their content. For basic presence systems, this content is a plain PIDF document carrying basic presence information. Instead of updating contact information as done for registrations, SIP publications are primarily focused on keeping the presence information (PIDF) of the presentity at the MPS up to date. Figure 4.3 shows an overview of the publication processes we implemented.

4.2.2 Implementation on Java Micro Edition

At first, we decided to implement all SIP processes (i.e registrations, publications, subscriptions) independently of each other. We grouped the operations together only later as discussed in section 4.4. We named the JME class that we created for managing publications as the `Publisher`. This class is shown in Figure 4.4. The `presence` variable holds the content to be published (the PIDF document). The `Presence` class is discussed in section 4.4.1.

As was done for the `RegistrationClient` class, the process of refreshing the publications is implemented in the `run` method. Instead of the using the `getCurrentAddress` method, however, the `getCurrentPresence` method is used to get the current PIDF document with the current presence information. The `getCurrentPresence` method returns a value of `null` if the presence content has not changed since the last publication or publication refresh. This method is then called in the `update` method of the `refreshHelper` instance of the `Publisher` class.

4.2.2.1 Java Micro Edition Request URIs

Initially when we started implementing SIP publications, the MPS was returning *403 Forbidden* responses to requests made with instances of the `Publisher` class. This problem resulted from the fact that the JME `SipClientConnection` instances always appends a port to the SIP request URIs. This seems to not be in keeping with the SIP standard as specified in [59].

At the MPS, a comparison between the SIP address in the request URI and the identity information (`entity` attribute of the `presence` element) of the PIDF document finds the discrepancy due to the incorrectly included port. To fix this problem, we considered modifying the JME SIP library itself. However the source code could not be found. Instead, we modified the SIP RA on Mobicents such that it always strips the port number from the request URI of all incoming requests, if present.

4.3 The Session Initiation Protocol Subscription Process

4.3.1 Session Initiation Protocol Subscription Message Flow and Requirements

With independent registration and publication processes completed, the next challenge involved implementing the JME `Subscriber` class for maintaining subscriptions to presence information. The requirements of the SIP subscription process are needed in understanding the next chapter which discusses SIP LBSs privacy. Therefore, we provide a detailed discussion of the subscription requirements here.

Header	Description
To	The URI of the resource at the server to which a subscription is being made.
From	The URI of the requesting LC.
Contact	The current contact address and port of the LC
Expires	The time for which a subscription dialog is valid at the server without refreshing
Event	The type of Event Package for which the subscription is made.
Accepts	The type of content accepted on the subscription dialog
Subscription-State	The current state of the subscription
Content-Type	The type of content within the body of the request (only for NOTIFY requests)
Content-Length	The length of the content in the body of the request (only for NOTIFY requests)

Table 4.2: SIP headers and values for subscription requests, adapted from [14]

The SIP headers for SIP SUBSCRIBE and NOTIFY requests are as shown and described in Table 4.2. Unlike SIP registrations and publications, SIP subscriptions need to be carefully maintained in SIP dialogs. A presence subscriber usually subscribes to multiple presentities simultaneously, necessitating multiple subscription dialogs [57, 71].The subscribing terminal receives notification requests from the MPS and it is important to identify the dialogs they belong to.

SIP messages in the same subscription dialog have the same value for the *etag* attribute in their *To* header. Subscriptions also have states, the most important of these states being *pending* and *active*. A *pending* subscription is unauthorised and yields no content in its notifications while, an active subscription is authorised and yields content in its subsequent notifications.

Figure 4.5 depicts the state diagram of a typical subscription dialog between `Subscriber` class instances and the MPS. Initial SIP SUBSCRIBE requests by `Subscriber` class instances cause the MPS to create a new SIP subscription dialog identified by unique *etag* value. From that point onwards, all the messages in the dialog must bear this *etag* value in their *To* header. The subscription state of the dialog is set to *pending* and remains so until the subscriber is authorised.

The MPS automatically authorises subscriptions for publishers that do not have presence privacy policy rules specified in the XDMS. Upon authorisation, the MPS sets the subscrip-

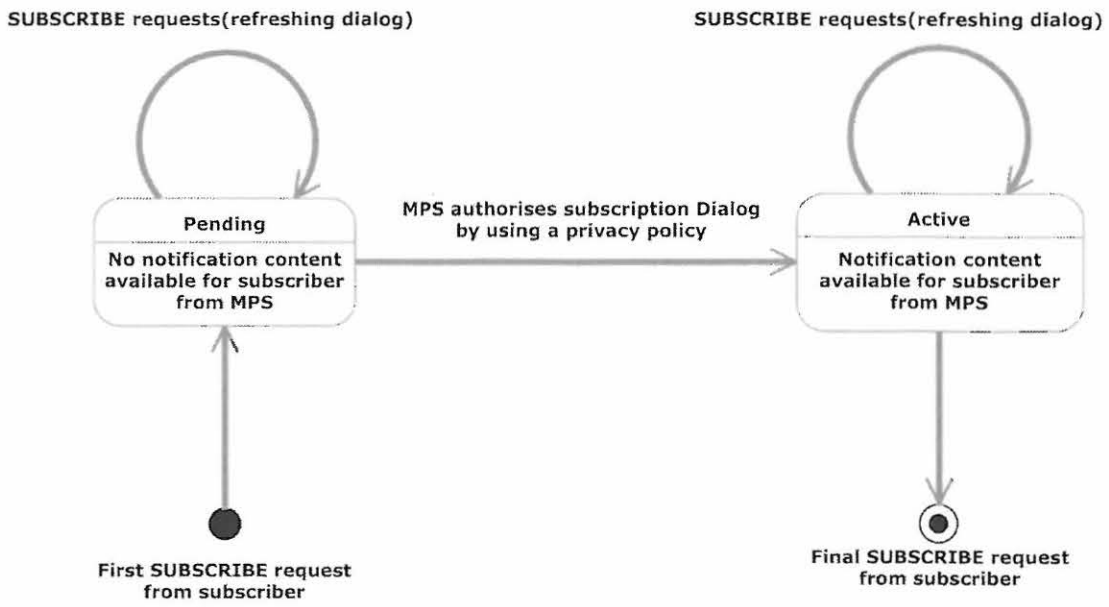

Figure 4.5: The SIP subscription dialog state diagram

tion state of the dialog to *active* and notifies the JME Subscriber with the PIDF document of the publisher. To end the dialog, the JME `Subscriber` sends a SIP SUBSCRIBE request with an *Expires* header value of 0.

4.3.2 Implementation on Java Micro Edition

The `Subscriber` class in Figure 4.6 works in the same way as the `Publisher` and `RegistrationClient` classes, except for a few minor differences in refreshing requests and processing incoming notifications. Each instance of the `Subscriber` class is responsible for only one subscription dialog to a single presentity. We introduce subscriptions to multiple presentities in the next section, when we re-organise the classes into an SDK.

4.3.2.1 Refreshing Subscriptions

JME `SipRefreshHelper` instances cannot be used to refresh subscriptions because they fail to maintain SIP subscription dialogs properly. Instead of keeping the value of the *etag* attribute of the *To* header, they store that of the *From* header. This causes the MPS to throw an exception because of failure to identify that a received subscription refresh

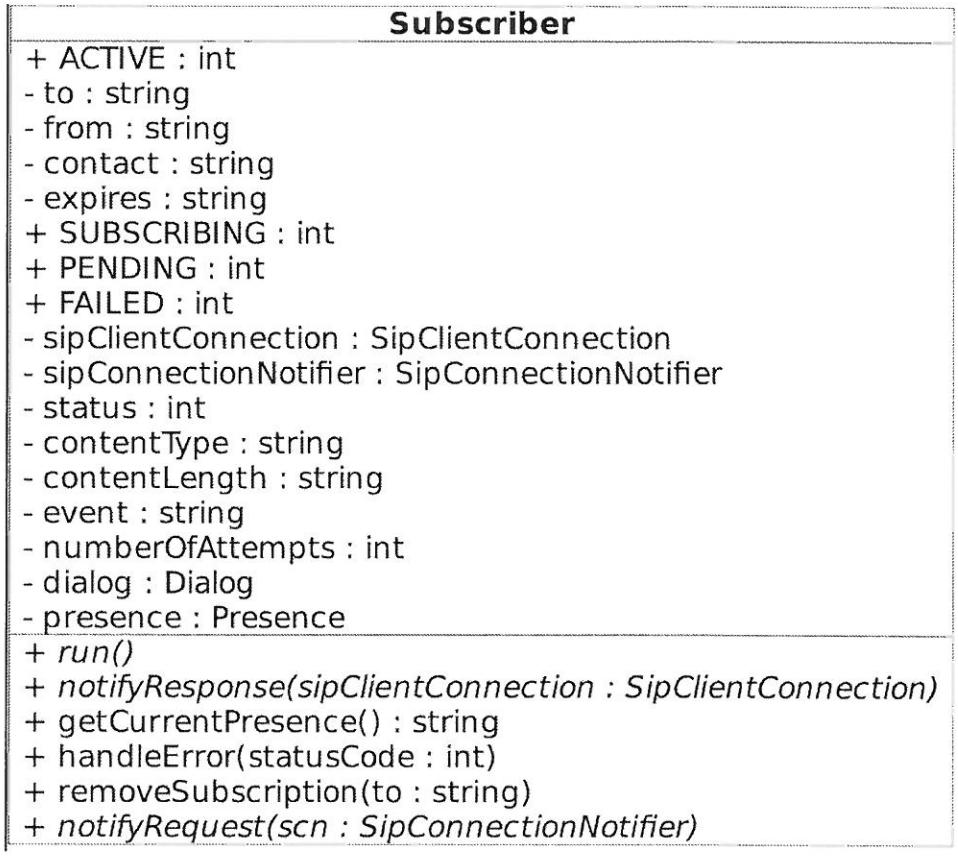

Figure 4.6: The `Subscriber` class

request is in the correct SIP dialog. We resorted to maintaining subscription dialogs by using JME `SipDialog` instances which keep track of the *etag* value of the *To* header.

The code snippet to refresh subscriptions is shown in Listing 4.4. Subscriptions are refreshed for as long as they are not failing (getting error responses) and while the subscription status is not *pending* for the first few subscription requests. The reason for this is that *pending* subscription dialogs bear no content and it is wasteful to keep maintaining such dialogs. We use the `SipDialog` instance's (`dialog`) `getNewClientConnection` method to create a connection for an already existing dialog. We also reset the *Expires* header to the expiry value suggested by the server. All the other headers remain unchanged.

```
1  public void run() {
2   while ((status != FAILED) && !((status == PENDING) && (
      numberOfAttempts > 5)) {
3
4    if (dialog != null) {
5    try{ scc = dialog.getNewClientConnection("SUBSCRIBE");
6    scc.setHeader("Expires", getExpires());
7    scc.send();
8    scc.close();
9    numberOfAttempts++;
10    }
11   if(status == CLOSING)
12     closeConnections();
13   Thread.currentThread().sleep(Integer.parseInt(getExpires())*1000)
       ;
14 } catch (Exception e) {
15 e.printStackTrace();
16 }
17 }
```

Listing 4.4: The run method code extract from the `Subscriber` class

4.3.2.2 Handling Session Initiation Protocol Notification Requests

The `Subscriber` class is responsible for handling notification requests made to SIP subscription dialogs. The class implements the JME `SipServerConnectionListener` interface (discussed in the Chapter 3) which requires the implementation of the `notifyRequest`

method. In the constructor of the `Subscriber` class, we create a JME `SipConnectionNotifier` instance and specify the port (5060) on which we listen for incoming connections. We also set the `Subscriber` as the listener for the requests incoming via its `SipConnectionNotifier` instance.

The notification request processing logic is located in the `notifyRequest` method (the handler method for incoming NOTIFY requests). Upon receipt of a SIP NOTIFY request, we check if the notification belongs to our subscription dialog and we extract the headers of the NOTIFY request and the content, if any is present. The status of `Subscriber` instances is set such that it reflects the value of the *Subscription-state* header of the NOTIFY requests in the `notifyRequest` method. We discuss the parsing of the notification content the (PIDF document) in section 4.5.1.2.

4.4 Re-organising the Session Initiation Protocol Functions into an Location-Based Services Development Kit Prototype

We have, so far, discussed the implementation of the `RegistrationClient`, `Publisher` and `Subscriber` classes for independent SIP registration, publication and subscription processes, respectively. These classes have many commonalities that we aimed to regroup more elegantly. Our objective to provide a toolkit environment on JME necessitated that we use a simple interface exposing the SIP processes of the LBSs SDK prototype (i.e registration, publication and subscription). This section discusses the process of re-organising the SIP processes so that they work simultaneously and can be invoked using simple methods.

4.4.1 The SipEvent and Presence Classes

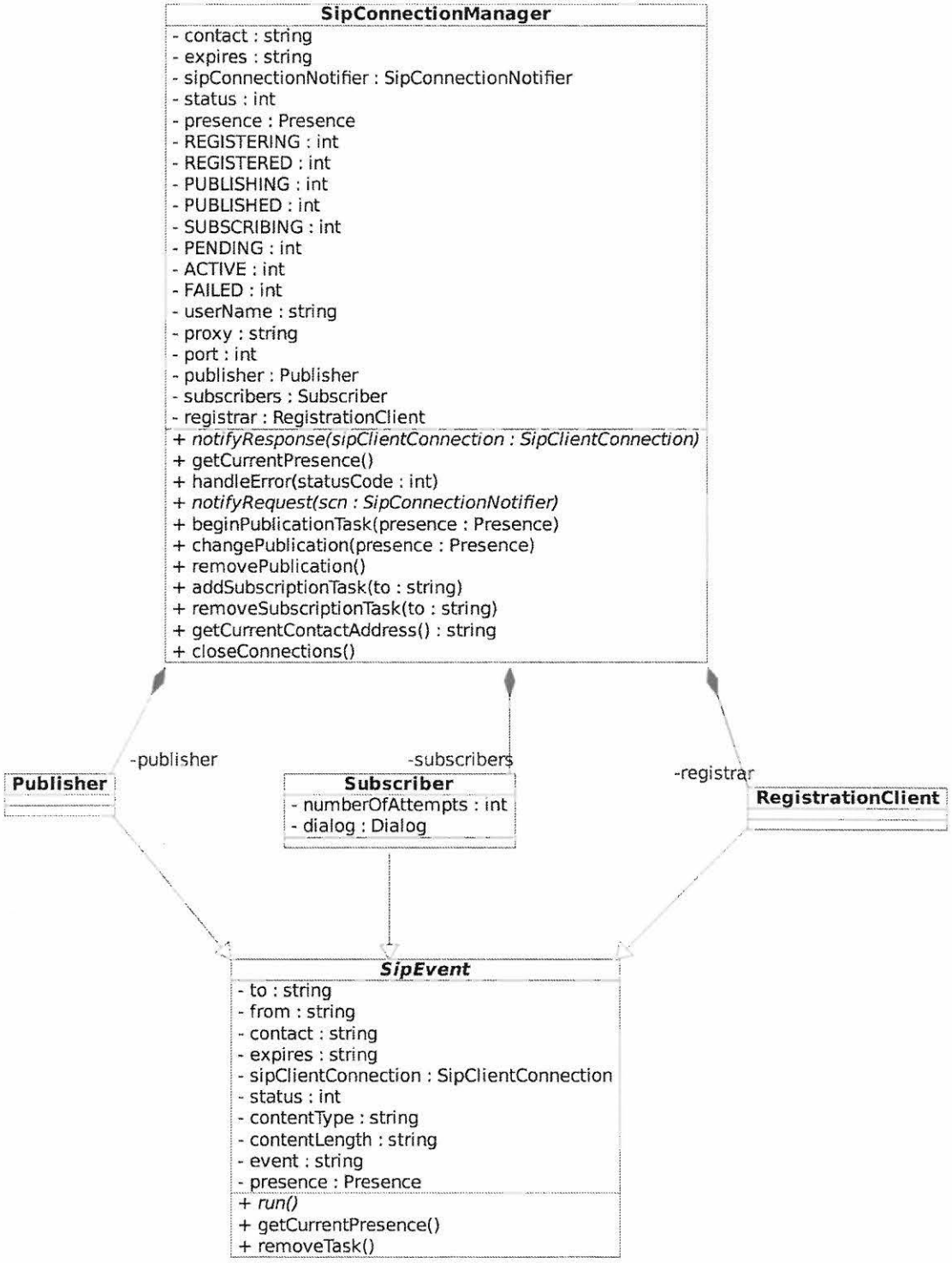

Figure 4.7: SIP classes in the re-organised prototype

Figure 4.7 shows the re-organised classes of the LBSs SDK prototype in the `org.rhodes.sipevent` package. Java documentation and source code of the toolkit have been provided with this thesis, we discuss only the important aspects of the classes here. One of the components shown in the figure is the `SipEvent` class. This is an abstract class we created for the common variables and operations of the `RegistrationClient`, `Publisher` and `Subscriber` classes. Most of the variables included in the class are those stored in the SIP headers of the SIP requests. The constructor of the class is used to set up the initial requests for the different SIP processes. The `removeTask` method contain the logic for performing the termination of the SIP processes.

Another important class of the LBSs SDK prototype is the `Presence` class in Figure 4.8. This class is contained within the `org.rhodes.sipevent.pojo` package. Instances of the class store the basic presence information of presentities namely the SIP address or identity of the presentity; the status, which is either *open* or *closed*; a note, which is a human-readable text about the presence information; and the time-stamp that bears the time when he document was composed (we ignore the location part for now). `Publisher` and `Subscriber` instances must own `Presence` instances so that they can store presence information. `Presence` instances store this information after it has been obtained from notification content, or before it is published.

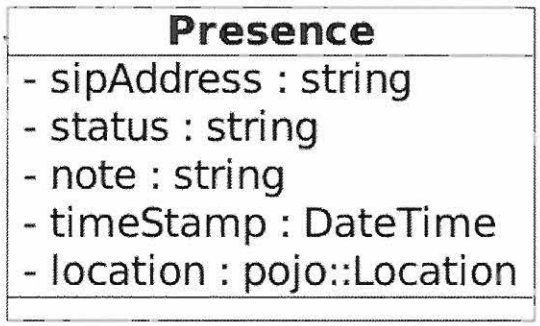

Figure 4.8: SIP classes in the re-organised prototype

4.4.2 The `SipConnectionManager` Class

Figure 4.7 shows the UML representation of the `SipConnectionManager` class of the `org.rhodes.sipevent` package. This class creates and controls the different SIP processes (i.e. registration, publication and subscription) by creating and managing `Reg-`

istrationClient, Publisher and Subscriber instances. It is designed to provide the methods for initializing and managing the SIP functionality provided by the LBSs SDK prototype. The SipConnectionManager class owns an instance of the Registrar class for making registrations to the Mobicents Proxy and Registrar Services. Registrations must be performed by all presence applications. If the registration operations fail, then no other tasks can be performed in the LBSs SDK prototype.

The SipConnectionManager class may have zero or one Publisher instances due to the fact that we limit the number of presentities or Targets per mobile device to one. It is also not always the case that a presentity wishes to share its presence information so it might not need to initiate publication tasks at all. The SipConnectionManager class may have zero or more Subscriber instances which are stored in a java.util.Vector data structure. Multiple Subscriber instances might be required because the watcher applications usually subscribe to multiple presentities or Targets simultaneously.

The getCurrentAddress method, formerly contained in the RegistrationClient class, is in the SipConnectionManager class so that all the other classes may use it for their *Contact* header values as well. The beginPublicationTask method can be used to start presence publications. The addSubscriptionTask method creates a new Subscriber instance to start a subscription dialog to a specified presentity or Target. The new Subscriber instance is added to the subscription Vector that keeps track of all the subscriptions at the mobile device. The removePublicationTask and the removeSubscriptionTask methods can be used to stop publication and subscription tasks, respectively. The getSubscriptionContent method returns a collection of the Presence instances of all the currently active subscriptions.

Most of these methods are aimed at exposing the SIP functionality of the prototype to LBSs developers. In Chapter 6, we discuss how such methods could be used to deliver simple centralised cross-referencing LBSs. Appendix A discusses them from a service developer's point of view to provide a panoramic view of our LBSs SDK prototype.

4.4.2.1 Implementing the SipClientConnectionListener and SipServerConnectioListener Interfaces

The SipConnectionManager class implements both the SipClientConnectionListener and SipServerConnectionListener interfaces formerly implemented by each one of the classes that handled SIP processes (i.e. RegistrationClient, Publisher and Subscriber).

The `SipConnectionManager` class, therefore, implements the `notifyResponse` and `notifyRequest` methods required by these interfaces. All the SIP responses to requests made by the mobile application are received and handled in the `notifyResponse` method. All the SIP requests addressed to the mobile application are received and handled in the `notifyRequest` method of the class.

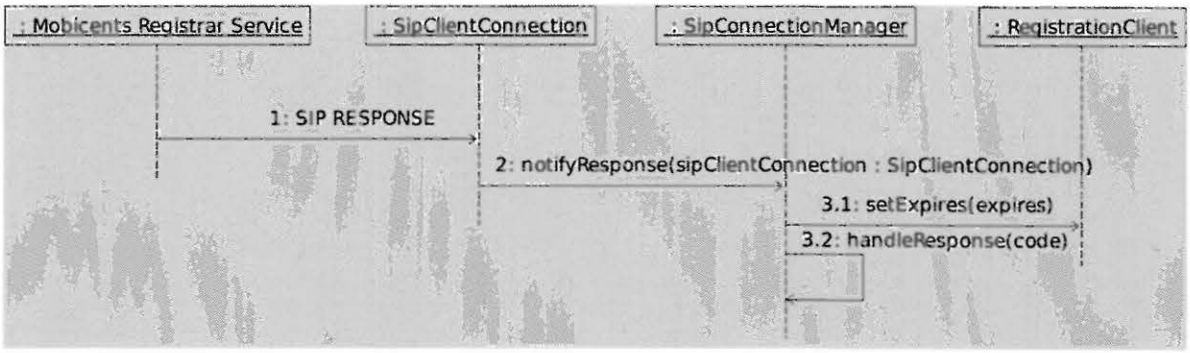

Figure 4.9: Handling a SIP registration response

The handling sequence of SIP registration responses in the LBSs SDK prototype is depicted in Figure 4.9. The Mobicents Registrar Service sends a response that is received by an instance of the `SipClientConnection` class at the SDK. The `notifyResponse` method of the `SipConnectionManager` class is invoked in step 2. The `notifyResponse` method determines the SIP method to which the SIP response is made (as shown in lines 7, 23 and 28 of Listing 4.5). The method then handles the parts of the response that are specific to the registration method. This includes extracting the value of the *Expires* header and passing it to the `RegistrationClient` instance (step 3.1 Figure 4.9) . The `getReg` method shown in the code snippet in Listing 4.5, returns the `RegistrationClient` instance of the `SipConnectionManager` class. This instance is used to set the properties of the SIP registration processes. Other SIP responses follow the same general sequence as the SIP registration response. The complete implementation of the `notifyResponse` method can be found in the source code provided with this thesis.

```
1 public void notifyResponse(SipClientConnection scc)
2 {
3 try {
4  scc.receive(0);
5  int responseCode = scc.getStatusCode();
```

```
6
7   if (scc.getMethod().equals("REGISTER"))
8   {
9    System.out.println("[SIPconnectionMannager]: Registration Response
       : " + scc.getStatusCode());
10   if (responseCode == 200)
11  {
12    getReg().setStatus(REGISTERED);
13    if (!scc.getHeader("Expires").equals(null))
14   {
15     getReg().setExpires(scc.getHeader("Expires"));
16   }
17  } else
18  {
19    getReg().setStatus(FAILED);
20    handleError(responseCode);
21   }
22
23  }if (scc.getMethod().equals("PUBLISH"))
24  {
25    //handle publication responses
26  }
27
28  if (scc.getMethod().equals("SUBSCRIBE"))
29  {
30    //handle subscription responses
31  }
32
33  } catch (IOException ex)
34  {
35  }
36  }
```

Listing 4.5: The `notifyResponse` method code extract from the `SipConnectionManager` class

The `notifyRequest` method is the same as it was implemented in the `Subscriber` class

earlier, except that it is designed to handle notifications from multiple subscription dialogs. Figure 4.10 depicts the general sequence that takes place whenever a SIP notification is made to the LBSs SDK prototype. The MPS sends a SIP NOTIFY request that might have content to the mobile device in step 1. Instances of the `SipConnectionNotifier` receive the request at the LBSs SDK prototype and invoke the `notifyRequest` method of the `SipConnectionManager` class.

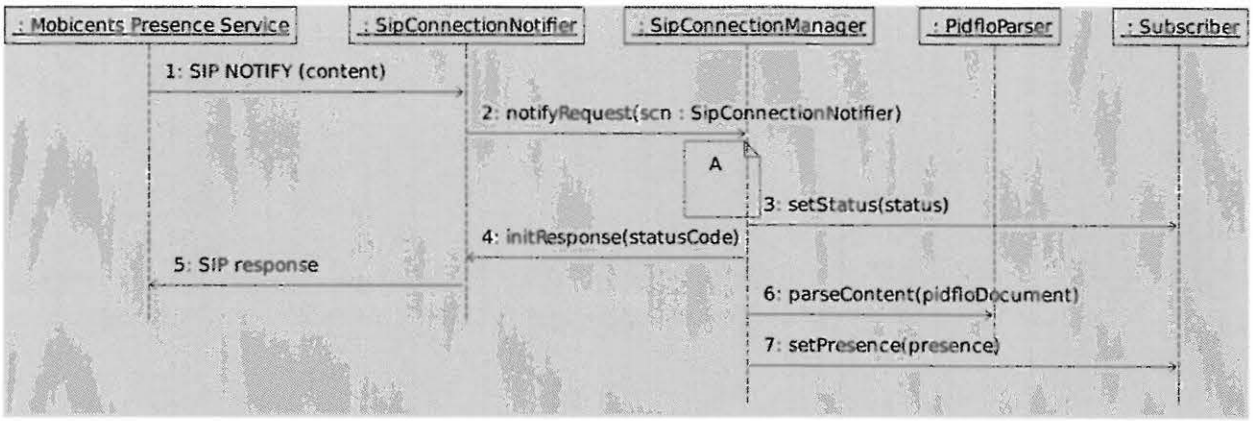

Figure 4.10: SIP notification request handling in the LBSs SDK prototype

The `SipConnectionManager` determines the subscription dialog to which the notification belongs by comparing the notification request dialog with that of *active* `Subscriber` instances at *A*. Once identified, the status of the appropriate `Subscriber` instance is set to reflect that of the subscription dialog in step 3. Step 6 invoke methods of the `PidfloParser` class to parse the PIDF document, if present. The presence information acquired is stored in the appropriate `Subscriber` instance in step 7. In step 4, the `SipConnectionManager` sends a SIP response to the MPS for the notification request.

4.4.2.2 Refreshing and Synchronising the Session Initiation Protocol Processes

Thus far, we have discussed how we re-organised the different SIP processes into the `SipEvent` and `SipConnectionManager` classes. We have not, however, discussed the synchronisation of these SIP processes. In SIP systems, SIP subcriptions and publications are usually preceded by SIP registrations. Our SDK is designed in such a way that if

registration processes fail, then the other processes are not performed because the mobile terminal has to be registered to use the services. The remaining paragraphs of this section focus on the synchronisation of the SIP processes.

In the LBSs SDK prototype, we first create a `RegistrationClient` class instance in the `SIPconnectionManager` class and start the registration thread. Thereafter, the developer using the finished LBSs SDK has an option of invoking the publication thread by calling the `beginPublicationTask` method (when implementing a Target entity). If an application needs LC functionality, then the `addSubscriptionTask` method can be invoked to start subscription threads. Once started, the threads are responsible for sending the initial requests and for periodically refreshing the SIP processes they represent for as long as they are needed.

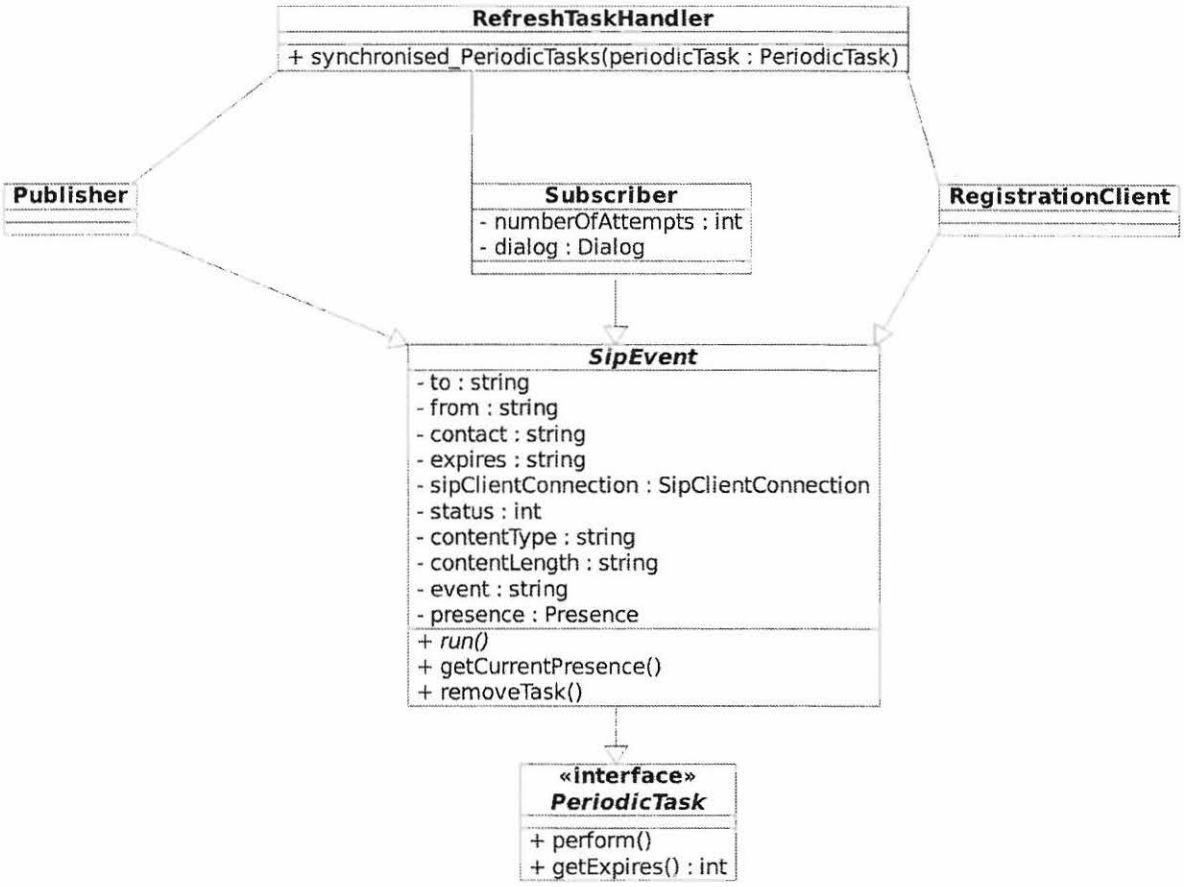

Figure 4.11: Syncronising the SIP processes in the `org.rhodes.sipevent` package

The `RegistrationClient`. `Publisher` and `Subscriber` classes all implement the `PeriodicTask` interface that we created. This is as depicted in the class diagram in Figure 4.11. The three classes implement the `perform` and the `getExpires` methods of the interface. The refresh operations of the different SIP tasks are implemented within the `perform` methods. The classes also all implement the abstract `run` method of the `Thread` class.

Within this `run` method is a while loop that generally runs for as long as the respective SIP task is required. Whether the loop continues running or not is determined by the conditions required by the individual SIP processes. For example registrations should be performed for as long as they are successful or awaiting an initial response from the server. The loop keeps invoking the synchronised `periodicTasks` method of the `RefreshTaskHandler` class that synchronises the processes depending on their expiry periods.

```
1 public void run()
2 {
3  while (getStatus() == SIPconnectionManager.REGISTERED || getStatus
     () == SIPconnectionManager.REGISTERING)
4   {
5     refreshTaskHandler.periodicTasks(this);
6   }
7 }
```

Listing 4.6: The run method code extract from the `SipConnectionManager` class

The `periodicTasks` method causes every `PeriodicTask` that has invoked it to wait for as long as its expiry period (value of the `expires` variable). The expiry period is expressed in milliseconds. After that period has elapsed, the method calls the `perform` method of the `PeriodicTask` that has invoked it. Finally, the method calls the `notifyAll` method of the thread class that causes other synchronised `PeriodicTask` instances to wake up. By using this process, we periodically refresh and synchronise the different SIP processes in the LBSs SDK prototype.

```
1 public synchronized void periodicTasks(PeriodicTask task)
2 {
3   try {
4         wait(Integer.parseInt(task.getExpires()) * 1000);
5         task.perform();
6         notifyAll();
7         } catch (InterruptedException ex)
```

```
8           {
9             ex.printStackTrace();
10           }
11  }
```

Listing 4.7: SIP task synchronisation in the `RefreshTaskHandler` Class

After developing an LBSs SDK prototype for presence applications, the next step involved making the both the JME LBSs SDK prototype and the MPS location-aware. As we explained in Chapter 2, we add Location Objects to PIDF documents for location information transport. Therefore, the step mostly involved enabling the creation, parsing and processing of Location Objects on both the JME LBSs SDK prototype and the MPS, as described in the following section.

4.5 Adding Location Object Processing to the Java Micro Edition Location-Based Services Development Kit Prototype and to the Mobicents Presence Service

4.5.1 Adding Location Object Processing Capabilities to the Location-Based Services Development Kit Prototype

While putting together the presence transport facilities of the LBSs SDK prototype in the last section, we introduced the `Presence` class that stores values pertaining to basic presence information. The `sipAddress`, and the `timeStamp` variables of the `Presence` class are useful dimensions of location information as mentioned in Chapter 2. The first step towards implementing support for Location Objects involved creating Plain Old Java Objects (POJOs) for storing location values.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <presence xmlns="urn:ietf:params:xml:ns:pidf"
3  xmlns:gp="urn:ietf:params:xml:ns:pidf:geopriv10"
4  xmlns:gml="urn:opengis:specification:gml:schema-xsd:feature:v3.0"
5  xmlns:gs="http://www.opengis.net/pidflo/1.0"
```

```
6  entity="pres:shange@sip.ru.ac.za">
7   <tuple id="vr45ur">
8    <status>
9     <basic>open</basic>
10
11 <-- Location Object starts here -->
12
13    <gp:geopriv>
14      <gp:location-info>
15       <gml:location>
16        <gs:Point gs:id="a" srsName="epsg:4326">
17          <gs:pos>25.78900 46.67834</gs:pos>
18        </gs:Point>
19       </gml:location>
20      </gp:location-info>
21    <gp:usage-rules>
22     <gp:retransmission-allowed>no</gp:retransmission-allowed>
23     <gp:retention-expiry>2010-09-23T06:35:35Z</gp:retention-expiry>
24    </gp:usage-rules>
25    </gp:geopriv>
26
27 <-- Location Object ends here -->
28
29  </status>
30    <timestamp>2010-09-22T06:35:35Z</timestamp>
31  </tuple>
32 </presence>
```

Listing 4.8: A sample PIDF-LO document

The XML snippet in Listing 4.8 shows a simple PIDF-LO document containing basic presence and a simple Location Object. Location Objects may be contained within the `geopriv` elements, which may be placed within the PIDF `status` elements. The `location` elements within the `location-info` elements contain the location information of the Target expressed using geodetic or civic format [49]. Since each Target is expected to be in one location at a given time, we introduce a constraint for our SDK:

- A single Target can publish only a Location Object describing one single discrete

location at a time. Civic or geodetic information may be contained within the same Location Object only for purposes of better describing a single discrete location.

4.5.1.1 Geodetic and Civic Information Plain Old Java Objects for the Location-Based Services Development Kit

For expressing geodetic information, the Location Object utilises a subset of GML (application schemas) customised for use in PIDF-LO documents [49]. We refer to these customised schemas as the Geoshape schemas. Geoshape schemas allow for two and three dimensional shapes to be expressed. To keep the toolkit simple, we implement only support for points, circles and polygons.

It can be seen from Listing 4.8 that a point constitutes a single pair of coordinates such as *37.23344 56.23467*. We create a `Point` POJO that contains variables to store the latitude and longitude coordinates. A `Circle` is expressed as a combination of a center point and a radius. A `Polygon` is expressed as a collection of points. The `Polygon` POJO contains one variable only, a vector of `Point` instances. The POJOs may be created and owned by a `Presence` instance. Since we cannot anticipate how the location information will be expressed by the Target, we created a Location interface which all POJOs implement. The resulting class diagram is as shown in Figure 4.12. All the location POJO classes are contained within the org.rhodes.sipevent.pojo package.

The civic location schema defined in [77] allows for location to be expressed with an optional seventeen fields ranging from the name of the country to more detailed levels such as seat numbers within a particular space. We decided to support only five of these fields: the `country` element; elements `A1` to `A4`, to hold national sub-division identifiers; and element `HNO`, to hold a house or street number. The `country` element must hold the alpha-2 codes from ISO 3166-1, such as *DE* for Germany and *ZA* for South Africa [77]. Elements `A1` to `A4` may contain national sub-divisions to better express the location of the Target. For instance, `A1` may be used to hold the value for the region or province and `A2` for a sub-division in a province such as a town. The `CivicAddress` POJO also implements the `Location` interface and contains a variable to keep a collection of civic addresses. This is also shown in the class diagram in Figure 4.12.

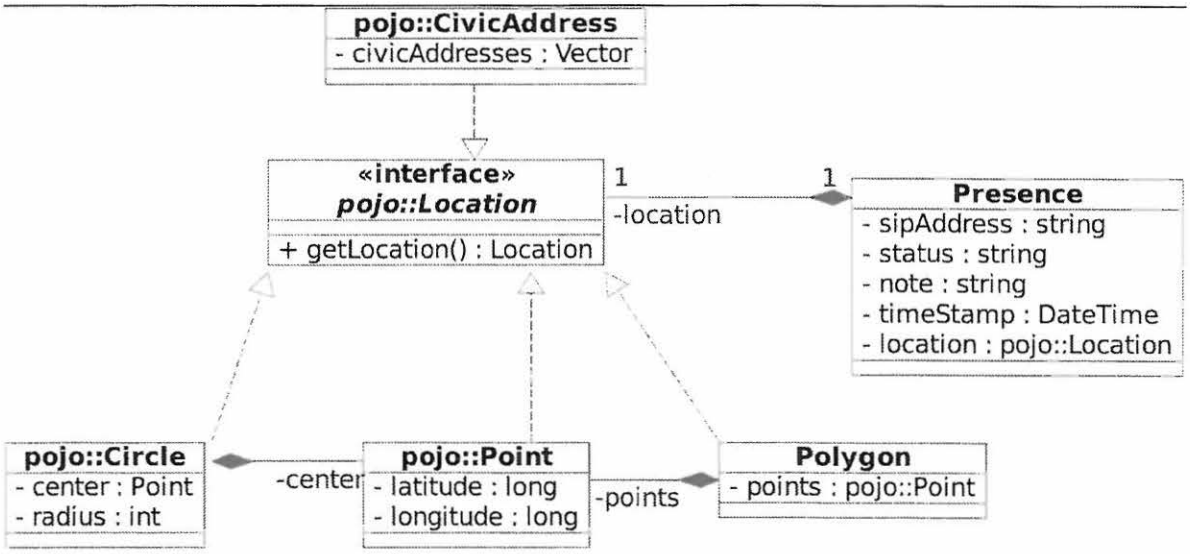

Figure 4.12: The classes of the org.rhodes.sipevent.pojo package

4.5.1.2 Geodetic and Civic Information Creation and Parsing

PIDF-LO documents in the LBSs SDK prototype are composed statically from strings and `Presence` instance variables. The parsing of the incoming PIDF-LO documents is performed using pull parsing. Pull parsing is a technique of parsing XML as a stream instead of building a tree (DOM) or pushing events out to client code (SAX) as it is done for push parsing [74]. The advantages of pull-based parsing as discussed in [74] include:

- a small memory footprint, because a whole tree of the document being parsed is not stored in memory;
- the parser parses only what it is required to parse, resulting in quicker performance than would be delivered by push parsing.

The pull parser we used for our implementation is known as kXML [74]. We created a class called `PidfloParser` that uses the kXML API to parse PIDF-LO documents. The static `parsePresence` method of the `PidfloParser` class is provided with a PIDF-LO String and it returns the resulting `Presence` POJO. This `parsePresence` method is used in the `notifyResponse` method of the `SipConnectionManager` class to parse the PIDF-LO document available during authorised SIP notifications.

4.5.1.3 Geodetic Location Determination with the Java Micro Edition Location API

We discussed the JME Location API in Chapter 3. The API can be used synchronously or asynchronously to obtain the location information of a mobile device, assuming that appropriate location determination technology such as GPS is available there [30].

We used the JME Location API for geodetic location information determination only. Civic locations may be provided by the users or developers through overloaded `beginPublicationTask` methods of the LBSs SDK prototype. These overloaded methods are discussed in greater detail in Chapter 6 and Appendix A. The `SipConnectionManager` class implements the `LocationListener` interface and implements the `locationUpdated` method. For the `SipConnectionManager` class to start publishing the location information from the mobile device location provider such as a GPS receiver, the developer must invoke an overloaded version of the `beginPublicationTask` method.

The overloaded `beginPublicationTask` method for automatic location determination takes in the `interval`, `timeout`, `maxAge` and `note` parameters. The first three parameters are provided to the `LocationProvider` instance's `setLocationListener` method. The use of the parameters is as described at [30]. The `note` parameter is for the human-readable presence information provided by the end user.

Every automatic invocation of the `locationUpdated` method causes the creation of a `Point` instance representing the current location of the mobile device. The original `beginPublicationTask` method is called to start publishing the newly acquired location information, together with the provided presence information. This is done only if the `SipConnectionManager` class is not already publishing other location and presence information. Otherwise, the `changePublication` method is invoked to change the published information instead. For more details on how this works, we refer the reader to the Java documentation and source code provided with this thesis.

4.5.2 Adding Location Object Processing to the Mobicents Presence Service

We have discussed the presence publication process shown in Figure 4.3, but have not discussed how the published information is processed at the MPS. In Chapter 3, we dis-

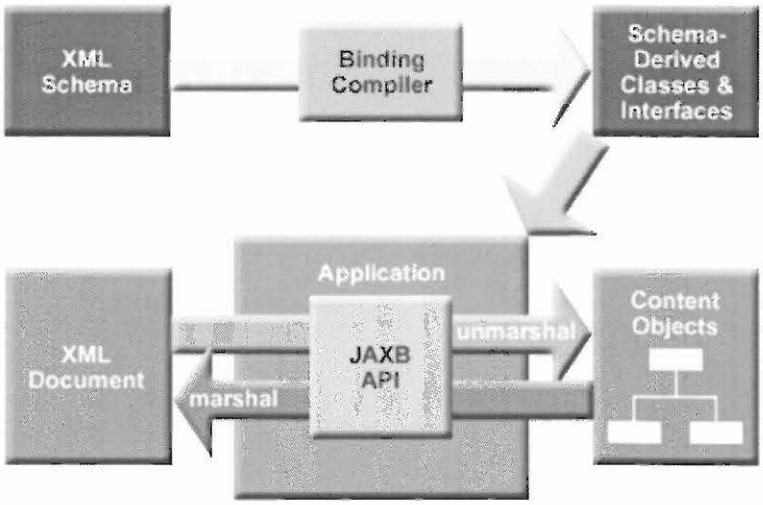

Figure 4.13: The JAXB tool overview [46]

cussed the MPS components and the SBBs of which it is composed. The presence information published in a PIDF document by a presentity is parsed and validated by the `PresencePublicationControlSBB` of the MPS. Another SBB of the MPS, the `PresenceSubscriptionControlSBB`, has the capability of composing PIDF documents used for SIP notifications.

The step towards adding location processing capabilities to the MPS involved modifying it to perform the validation, parsing and creation of Location Objects that are part of the PIDF-LO documents. The tool used for XML document validation, parsing and creation on the MPS is the Java Architecture for XML Binding (JAXB) [46].

4.5.2.1 Location Object Schema Binding

Shown in Figure 4.13, the JAXB tool allows for the easy and quick conversion of an XML Schema into corresponding POJOs. This process is called binding and it is performed by the binding compiler [46]. We bound all the Location Object schemas to the equivalent POJOs. The Location Object JAXB POJOs that were generated were placed in the same package as the JAXB POJOs originally generated for parsing and composing presence information (PIDF) on the MPS. The reason for co-locating these POJOs was that they are all used to parse and compose PIDF-LO documents.

4.5.2.2 Location Object Processing in Service Building Blocks

The second part of Figure 4.13 shows that the JAXB API can be used to marshal and unmarshal XML documents. The process of unmarshalling a document involves converting the XML contents to an equivalent Java Object Content Tree (instantiated from the JAXB-generated POJOs). Marshalling a document involves the reverse process to unmarshalling. It converts a Java Object Content Tree to the corresponding XML document. We added the newly-generated Location Object POJOs to the JAXB Contexts of the `PresencePublicationControlSBB` and the `PresenceSubscriptionControlSBB` so they can be used through the JAXB API to parse and create PIDF-LO data.

The `PresencePublicationControlSBB` has a helper class for composing presence publications. Presence composition is specified in [14] and it is necessary when the presence state of a presentity is derived from different UAs. In our case, a Target can only be at one location at a time and no location composition is required. For this reason, we make sure that when a new publication is made, the old Location Object data is overwritten with the newly published data.

The `PresenceSubscriptionControlSBB` of the MPS uses only the published data for composing PIDF-LO documents provided by the `PresencePublicationControlSBB`. No modification was required of the `PresenceSubscriptionControlSBB` to enable Location Object processing. In the next chapter, however, we modify the `PresenceSubscriptionControlSBB` to enable location-based privacy.

4.6 Chapter Summary

In this chapter, we have discussed the initial steps taken during the implementation of the cross-referencing LBSs SDK prototype and the modification made to the MPS. In the preceding chapters, we made the decision to reuse the MPS presence signaling system to transmit location information. We started with building a presence prototype that works with the MPS. We discussed the requirements of the three SIP processes necessary for the realisation of the prototype. These processes involved maintaining SIP registrations, publications and subscriptions. The SIP processes were first implemented independently of each other and were later re-grouped into an LBSs SDK prototype.

With the completion of the SIP LBSs transport facilities of the LBSs SDK prototype,

we introduced a location dimension to the LBSs SDK prototype by enabling Location Object information processing. We created POJOs to hold the location information that was obtained or written to the Location Object documents. For our SDK, the location information is expressed with a limited subset of shapes and civic locations. Future work in this area can focus on adding capabilities to process more shapes and civic locations. Finally, we introduced Location Object processing within the MPS by making the necessary modifications there.

Chapter 5

Adding Privacy Support

The preceding chapter focused on developing a JME LBSs SDK prototype for presence and location transmission, and on modifying the MPS so that it processes PIDF-LO documents. In this chapter, we discuss the addition of location-based privacy to both the JME LBSs SDK prototype and the modified MPS. We explained in Chapter 2 that we aim to provide location-sharing privacy through the use of geolocation policies. (Geolocation policies provide Targets with means to control the sharing of their Location Objects.) Figure 5.1 depicts the privacy architecture implemented as part of our work. A Target or a separate rule-making entity can PUT, GET and DELETE geolocation policy documents at the XDMS. Different LCs, subscribing to the PIDF-LO document of the Target, are then given different versions of the document by the MPS according to the geolocation policy document rules of the Target. The protocol for geolocation policy communication between the XDMS and Targets or separate rule-making entities is XCAP. In Chapter 3, we mentioned that we could not find a JME implementation that supports this protocol. For this reason, we had to create our JME XCAP implementation for our JME LBSs SDK.

Geolocation-policies play an important role in assuring privacy in the sharing of a Target's location. When a Target publishes its location information to the network, this information could be used and shared with a wide range of LC applications. These LC applications could be hosted by marketing organisations, government agencies, emergency service providers or on end-user mobile phones. Geolocation-policies give a Target the means to specify how their location information should be shared with the LCs that are authorised to see it. Without these policies, every LC will get the same information which might not always be appropriate.

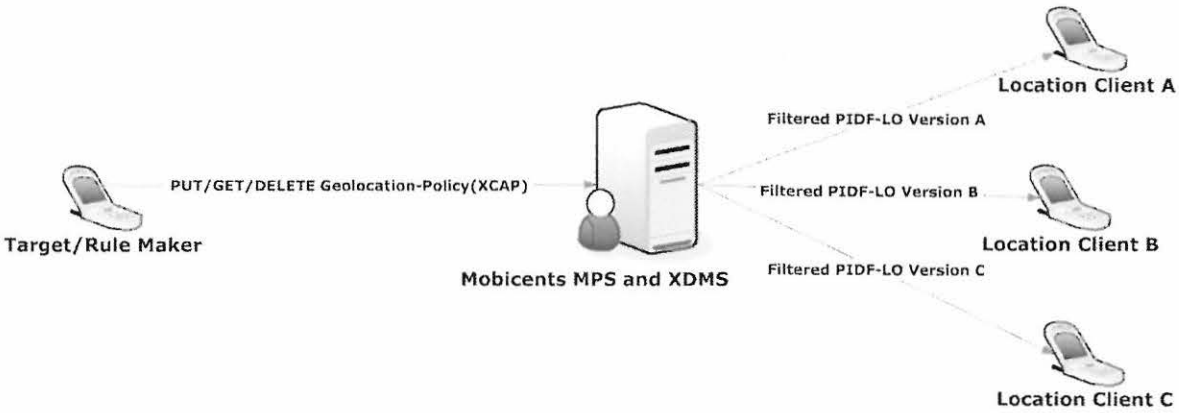

Figure 5.1: The geolocation policy privacy architecture, adapted from [66]

The contents of this chapter are better understood with some background regarding the format of the geolocation-policy document. We provide this background first. Thereafter, we focus on adding the geolocation-policy application usage (appusage) to the Mobicents XDMS. The third section discusses adding XCAP and geolocation-policy support to our JME LBSs SDK prototype. The fourth section discusses the final piece of the puzzle, which is the implementation of privacy-aware location subscription at the modified MPS, using geolocation-policy document rules. Section five then provides a summary of the chapter.

5.1 The Geolocation-Policy Document

5.1.1 Common-Policy for Location-Based Services Privacy

Listing 5.1 shows a sample geolocation-policy document that a Target might own and maintain at the XDMS. Geolocation-policy extends the IETF common-policy and inherits all of the common-policy properties. Common-policy defines three extensible parts of authorization rules. These are conditions, actions and transformations. A *rule* is analogous to a conditional statement used for programming. The conditions could be viewed as the *if* part, and the actions and transformations as the *then* part [67]. A privacy policy may contain one or many of these authorization rules (represented with `rule` elements in Listing 5.1) [67].

The important conditions defined for common-policy include the `identity` and `validity` elements (conditions) which are in the `conditions` element in Listing 5.1. The `identity`

condition may have one or more `one` and `many` elements containing the addresses or domains of the LCs that may be affected by the rule [67]. The `validity` condition specifies the time frame for which the rule is valid. Actions and transformations are applied only if all the conditions resolve to true when compared with the current state of the subscription and the published PIDF-LO document. They are specified within geolocation-policy schemas because they are specific to the data for which authorisation is being provided (such as a Location Object).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cp:ruleset
3  "xmlns:gp="urn:ietf:params:xml:ns:geolocation-policy"
4  "xmlns:cp="urn:ietf:params:xml:ns:common-policy"
5  "xmlns:ca="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddrr"
6  "xmlns:gml="http://www.opengis.net/gml"
7  "xmlns:lp="urn:ietf:params:xml:ns:basic-location-profiles"
8  "xmlns:gs="urn:ietf:params:xml:ns:pidf:geopriv10:geoShape">
9  <cp:rule id= "closefriends">
10  <cp:conditions>
11   <cp:identity>
12    <cp:one id="sip:shange@sip.ru.ac.za"/>
13    <cp:many domain="convergence.sip.ru.ac.za"/>
14   </cp:identity>
15   <cp:validity>
16    <cp:from>2010-06-24T17:00:00+01:00</cp:from>
17    <cp:until>2010-07-24T19:00:00+01:00</cp:until>
18   </cp:validity>
19   <gp:location-condition>
20    <gp:location  profile="civic-condition"
21     xmlns="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr">
22     <civicAddress>
23      <country>ZA</country>
24      <A1>Eastern Cape</A1>
25      <A3>Grahamstown</A3>
26     </civicAddress>
27    </gp:location>
28    <gp:location  profile="geodetic-condition">
29     <gs:Circle srsName="urn:ogc:def:crs:EPSG::4326">
```

```
30        <gs:pos>42.5463 -73.2512</gs:pos>
31        <gs:radius uom="urn:ogc:def:uom:EPSG::9001">850.24</gs:radius>
32       </gs:Circle>
33      </gp:location>
34     </gp:location-condition>
35    </cp:conditions>
36    <cp:actions>
37      <gp:sub-handling>allow</gp:sub-handling>
38    </cp:actions>
39    <cp:transformations>
40     <gp:provide-location profile="civic-transformation">
41       <lp:provide-civic>country</lp:provide-civic>
42     </gp:provide-location>
43     <gp:provide-location profile="geodetic-transformation">
44       <lp:provide-geo radius="500"/>
45     </gp:provide-location>
46    </cp:transformations>
47  </cp:rule>
48  </cp:ruleset>
```

Listing 5.1: A sample geolocation-policy document

5.1.2 Geolocation-Policy

The geolocation-policy schema introduces the `location-condition` element into the `conditions` element to specify the location-specific conditions of the Target. The `location-condition` element may contain one or more `location` elements each describing a discrete location within which a Target should be located for the `location` element(condition) to evaluate to true. As can be seen in Listing 5.1, the discrete location within the `location` elements can be expressed either using human-readable addresses or geodetically. If expressed geodetically, a Geoshape circle is used to describe the area within which the Target should be located. If expressed using human-readable addresses, the civic elements that best describe the area where a Target should be located are used. The `location` elements are evaluated using a logical *OR* within the `location-condition` element, so at least one of the `location` elements must evaluate to true for the `location-condition` element to be true [66].

The `actions` element of the rule determines the action that should be performed if the `conditions` evaluate to true. These actions are either `block` or `allow`, resulting in an LC receiving no information at all if blocked or receiving information that has transformations applied to it otherwise. In the `transformations` element, the Target can specify the granularity of civic and/or geodetic information that should be provided to the LCs affected by the rule to ensure location privacy. If expressed civically (within the `provide-civic` elements), the level of the civic element that should be provided from the PIDF-LO is specified. If expressed geodetically, (within the `provide-geo` elements), a radius is given to introduce a deliberate error to the geodetic location expressed within the PIDF-LO document [66].

5.2 Adding the Geolocation-Policy Appusage to the Mobicents XML Document Manangement Server

As we have discussed in Chapter 2, privacy policies are kept and maintained at the XDMS. The Mobicents XDMS supports presence rules for providing privacy to presence applications [71],but it does not support geolocation-policy. In order for the Targets to be able to create, modify and delete geolocation-policy documents at the Mobicents XDMS, we need to add a geolocation-policy appusage first. As mentioned in [34], the main function of the appusage is to provide to the XDMS the following information:

- The XML schemas that define the structure and constraints of the geolocation-policy data.
- An appusage identifier (*auid*) for geolocation-policy documents.
- The MIME type of geolocation-policy data.
- The keys to be used in the URIs to facilitate client (i.e. Target, MPS.) access to the geolocation-policy data.

Presence privacy policies do not affect the basic presence information that we have decided to transmit as a part of our implementation. Instead, they affect information provided in PIDF document extensions such as the presence Data Model [62] and Rich Presence Information Data Format (RPID) [65]. Our focus is on location: therefore we leave the

basic presence information to be provided to the authorised LCs without it being affected by the privacy policy filtering.

5.2.1 The Required Schemas and Java Architecture for XML Binding Components

The first step to creating the geolocation-policy appusage involved collecting the necessary schemas and putting them in a place where they could be loaded and used by the `GeolocationPolicyAppUsage` class (discussed in 5.2.2.1) [34]. The `GeolocationPolicyAppUsage` class accepts a reference to one default namespace only. The other namespaces must be directly or transitively referenced in the schema of the default namespace. In the geolocation-policy appusage, the geolocation-policy namespace (`urn:ietf:params:xml:ns:geolocation-policy`) is the default namespace. We replaced the XML schema `any` elements with reference to elements from the appropriate namespaces. The elements that we directly or transitively referenced from the geolocation-policy schema are from the following namespaces [66]:

- `urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr`
- `urn:ietf:params:xml:ns:pidf:geopriv10:geoShape`
- `http://www.opengis.net/gml`
- `urn:ietf:params:xml:ns:basic-location-profiles`

Multiple schemas from the GML namespace, `http://www.opengis.net/gml`, use XML schema `include` elements instead of XML schema `import` elements to reference each other using a single namespace. We discovered that the Mobicents SchemaContext class that combines multiple schemas does not support XML schema `include` elements, only `import` elements as discussed in [36]. Since the schemas use the same namespace, we made a decision to combine them all in one schema. The disadvantage of this approach is that it could be inefficient for large scale deployments, a situation that could be addressed in future by modifying the `SchemaContext` class to process `import` elements.

After collecting and correctly associating the different schemas, JAXB bindings for all the schemas needed to be created. The reason for this is that the data expressed in the policies

needs to be validated, parsed and made available to the MPS `PresenceSubscriptionControlSBB` so that it may be used to filter the Location Objects for location subscriptions. The process of creating and using JAXB bindings is the same process that has been discussed in 4.5.2.1.

5.2.2 Using the Mobicents XML Document Management Server Interfaces and Abstract Classes to Add the Appusage

The Mobicents XDMS provides interfaces and abstract classes for the easy addition of new appusages [34]. In this section, we discuss the use of these facilities to implement the geolocation-policy appusage.

5.2.2.1 The XML Document Management Server `GeolocationPolicyAppUsage` Class

Each Mobicents XDMS appusage is represented by a class extending the `org.openxdm.xcap.common.appusage.AppUsage` abstract class [34]. We created such a class for the appusage and called it the `GeolocationPolicyAppUsage` class. In this class, we specify, as required:

- the default schema namespace: `urn:ietf:params:xml:ns:geolocation-policy`;
- our appusage identifier (auid): `geolocation-policy`;
- and the MIME type of our documents: `application/auth-policy+xml`.

The `org.openxdm.xcap.common.appusage.AppUsage` abstract class provides methods for specifying data constraints and appusage inter-dependencies. There is no specific interplay between the geolocation-policy appusage and other appusages such as that of presence policies. For this reason, we do not override these methods. Additional constructors of the abstract class allow for custom authorization schemes to be specified. We use the constructor that provides us with default authorization (i.e a Target can read, write or delete only their own geolocation-policy documents).

5.2.2.2 The XML Document Management Server `GeoLocationAppUsageFactory` Class

In addition to the `GeolocationPolicyAppUsage` class, we also implemented the org.openxdm.`xcap.common.appusage.AppUsageFactory` interface [34]. Implementing this interface ensures that the XDMS maintains a cache of the appusage schemas and document validation objects so that they are not recreated after every invocation. We named this class `GeoLocationAppUsageFactory`.

5.2.2.3 The XML Document Management Server `GeoLocationAppUsageSbb` Class

Finally, we extended the `org.openxdm.xcap.server.slee.abstractAppUsageSbb` abstract class with a class we named `GeoLocationAppUsageSbb` (SBB) [34]. This SBB uses the `GeolocationPolicyAppUsage` and `GeoLocationAppUsageFactory` classes to load and unload the geolocation appusage objects into the Mobicents XDMS. The SBB loads the geolocation-policy schemas from the directory where they are stored and combines them into one using the default schema namespace. It then provides them to an instance of the `GeolocationPolicyAppUsage` class. The overall function of the SBB is to allow the SLEE to load and unload the geolocation-policy appusage into the Mobicents XDMS.

5.2.2.4 Specifying Keys For Access URIs

As we mentioned in Chapter 2, the protocol used by the Targets to create, modify and delete documents is known as XCAP. XCAP is based on HTTP and uses URIs to identify the documents, elements or attributes that are to be created, modified or deleted. To complete the creation of the geolocation-policy appusage, we provided a *key* to be used in the construction of the access URIs. These is done by extending the following abstract classes:

- `org.openxdm.xcap.common.key.UserDocumentUriKey`
- `org.openxdm.xcap.common.key.UserElementUriKey`
- `org.openxdm.xcap.common.key.UserAttributeUriKey`

We extended these classes with the `GeoLocationPolicyUserDocumentUriKey`, the `GeoLocationPolicyUserElementUriKey` and the `GeoLocationPolicyUserAttributeUriKey`, respectively. We call the parent constructors of each one of these classes and provide the *auid*, `geolocation-policy`, as the *key*. This step completes the addition of the geo-location-policy appusage to the Mobicents XDMS.

5.3 Adding Privacy Facilities to the Location-Based Services Development Kit Prototype

Having added the geolocation-policy appusage to the Mobicents XDMS, the next step involved creating facilities for the client environment, the JME LBSs SDK prototype, to use the appusage. Figure 5.1 shows that a Target or separate rule-making entity must use the XCAP protocol to perform operations at the XDMS. The XCAP protocol is a specialised application of HTTP for remotely modifying appusage resources. In this section, we evolve a simple HTTP client into an XCAP client capable of performing operations on remote geolocation-policy documents.

Before proceeding any further, it is important that the reader understands what an XCAP resource is: *"An HTTP resource representing an XML document, an element within an XML document, or an attribute of an element within an XML document that follows the naming and validation constraints of XCAP"* [34].

The reader also needs to understand how XCAP resources are organised at the XDMS. Figure 5.2 shows that the XCAP resources stored on the XDMS for different appusages and users are logically organised into a tree structure. The root of the tree is referred to as the XCAP Root and it contains all the documents managed by the XDMS for different users and appusages. Each appusage sub-tree may have a global sub-tree only; a sub-tree containing user directories only; or a sub-tree containing both global and user sub-trees. Global sub-trees contain XCAP resources that are accessible by all users without any authentication. The users' directory sub-trees, on the other hand, contain sub-trees belonging to individual XCAP users. Each user's directory may contain the XCAP resources that are private and accessible only to them.

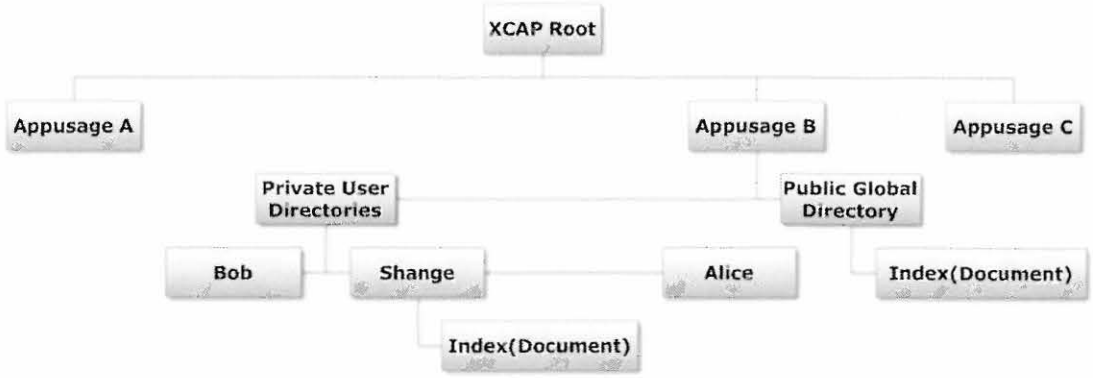

Figure 5.2: The structure of XML resources on the XDMS, adapted from [23]

5.3.1 Adding Geolocation-Policy Document Operations Support

The geolocation-policy appusage has a tree for user's directories and no global sub-tree. Users have to create profiles with user names and passwords before they can invoke requests appusages at the XDMS. The Mobicents XDMS provides a web-based interface for registering such profiles [55]. This interface is further discussed in Appendix A. Digest authentication is used for authorising XCAP requests to ensure that the users' directories are secure, and that each user can read, write and modify only their own XCAP resources.

5.3.1.1 Preparing the HTTP Client for XML Configuration Access Protocol Support

We mentioned in section 3.2.3 that we were unable to find a reusable implementation of an XCAP client on JME. For this reason we had to add XCAP functionality on top of an existing HTTP client. We started by downloading a simple JME HTTP client class from [20] that was designed for Representational State Transfer (REST) web services. The class used JME `HttpConnection` instances to perform HTTP POST and GET requests. It also had the capability of handling basic HTTP authentication and HTTP errors. We renamed this class and called it the `HttpClient`.

While trying to perform experimental geolocation-policy document operations, we realised that the HTTP stack for JME MIDP 2.0 does not support HTTP PUT and DELETE methods which are necessary for XCAP operations [76]. The only supported methods are the HTTP HEAD, POST and GET methods. No particular reason has been given for this

omission. To solve the problem, we took into consideration modifying and recompiling the JME SDK source code. However, the source code was not available, as was mentioned in the previous chapter.

We later chose to emulate HTTP PUT and DELETE requests by using the HTTP POST requests which are already supported in the JME HTTP stack. HTTP POST requests are not used in XCAP. As specified in IETF RFC 2616, an HTTP POST request may contain a payload and a URI specifying the server resource that should process that payload. A PUT request is semantically similar to a POST request but a URI in a PUT request specifies where the payload must be stored for future access. We emulate PUT requests on JME by putting storage URIs and content into HTTP POST requests. HTTP DELETE requests require a URI of the resource to be deleted from a particular location to be communicated to an HTTP server. Therefore we used HTTP POST requests with no content to emulate HTTP DELETE requests. Modifications needed to be made on the Mobicents XDMS to complete the emulation.

Since the XCAP protocol does not support HTTP POST requests, the Mobicents XDMS was altered in such a way that incoming POST requests are treated either as PUT or DELETE requests. If a POST request contains content, its content and URI are passed to an HTTP PUT request handler, otherwise the URI is forwarded to the HTTP DELETE request handler. This is elaborated more in section 5.3.1.3 and it is illustrated in Figure 5.3. This workaround has a shortcoming in that the JME LBSs SDK prototype becomes inflexible as every XDMS has to be modified to deal with the modification.

A better solution involving the use of low-level sockets could be used instead of our emulation in future. An HTTP stack for incoming and outgoing HTTP messages will then need to be implemented on JME. However one should be aware that as specified in [31]: "*MIDP 2.0 implementers are required to provide support for HTTP 1.1 communication, but support for low-level TCP/IP sockets and UDP/IP datagrams is optional*". This means that applications that utilise custom sockets may not be universally supported by all devices that run JME.

5.3.1.2 Providing Digest Authentication for XML Configuration Access Protocol Authorisation

The Mobicents XDMS uses digest authentication to authorise requests to private user directories. The authentication key used is generated using a Message-Digest Algorithm

5 (MD5) cryptographic hash function. We modified the `HttpClient` class such that it uses MD5 instead of basic authentication. For generating the MD5 cryptographic key, we downloaded and modified Java classes obtained from [47].

5.3.1.3 Experimental XML Configuration Access Protocol Document Operation Support Using the HTTP Client

We put together the `HttpClient` and authentication classes to perform XCAP document and element operations. In this sub-section, we provide a discussion only on the XCAP document operations. The URI format used by XCAP clients to identify XCAP resources in the user directory sub-tree of any appusage is:

http://{xcap-server-address:port}/{xcap-root}/{auid}/users/{user name}/index.

The Mobicents XDMS listens for XCAP requests on port 8080, and the XCAP Root is *mobicents*. Replacing this in the URI format gives the URI:

http://server_address:8080/mobicents/geolocation-policy/users/{username}/index.

This is the URI used to PUT, GET and DELETE geolocation-policy documents.

Figure 5.3 shows the mobile device performing preliminary document operations on the XDMS. The `GeolocationPolicyAppUsage` class, an experimental class that we created on JME, was used to perform these operations. The class uses the `HttpClient` class to invoke geolocation-policy document requests from the Mobicents XDMS. The exchange of messages between the entities is simplified to highlight only the important aspects. In step 1.1, the `GeolocationPolicyAppUsage` class invokes the `HttpClient` class to send an authenticated POST request. This request carries the document URI and bears the geolocation-policy document in its body (Step 1.2). At the XDMS, the POST request is checked for content and it is subsequently treated as a PUT request. The XDMS takes the content (the geolocation-policy document) and puts it in the user's directory. A 201 Created response is sent to the `HttpClient` class indicating the success of the operation in step 1.3.

Step 2.1 in Figure 5.3 depicts another independent scenario where the `GeolocationPolicyAppusage` class invokes the `HttpClient` to make a POST request to the Mobicents XDMS. The difference between this request and that made in step 1.2 is that this request does not bear any content. At the XDMS, the absence of content in the body of the request causes it to be treated as a DELETE request. This causes the XDMS to delete the

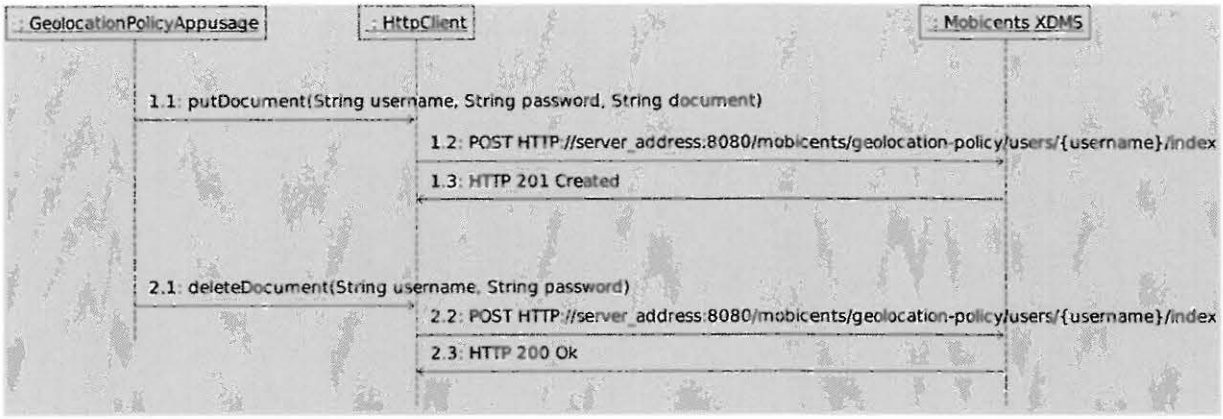

Figure 5.3: Geolocation-policy XCAP document operations

geolocation-policy document of the user if it exists there. The XDMS sends an HTTP 200 OK response upon the success of the operation (step 2.3). HTTP GET requests with the document URI cause the XDMS to return the content (geolocation-policy document) to the client if it exists at the XDMS (not shown in the Figure). This completes the discussion of the experimental geolocation-policy XCAP operations at document level. In the next section, we discuss the geolocation-policy element operations.

5.3.2 Adding Geolocation-Policy Element Support

Geolocation-policy document elements, by definition, are XCAP resources as well and the `GeolocationPolicyAppUsage` class on the mobile device should be provided with facilities to manipulate them. This is necessary to allow LBSs entities to fine-tune the geolocation-policy document to the specific requirements of the users or applications. When adding, deleting or modifying elements of XCAP documents, the requirement is that a correct URI that references the element is used. In the case of element addition and modification, the element to be added in the document at the XDMS is carried within an HTTP PUT request body. If the request results in a document that violates the structure given in the geolocation-policy document schemas, then an error (409 conflict) is sent by the XDMS in response [23]. To understand properly how the XCAP client was put together, one has to understand the semantics of constructing XCAP element URIs.

5.3.2.1 URI Semantics for Element Operations

The URI format for XCAP element manipulation on mobicents is:

http://server_address:8080/mobicents /auid/users/user_Id/index/~~/node-selector.

The *node-selector* contains the path to the particular element within the document being modified [23]. To explain this further, we use the geolocation-policy document sample shown in the XML snippet in Listing 5.1. To access the *<cp:one id="sip:shange @146.231.123.109"/>* element in line 12, we use the URI shown in Listing 5.2:

```
1 http://server_address:8080/mobicents/geolocation-policy/users/
    user_id/index/~~
2 /cp:ruleset/cp:rule%5b@id=%22closefriends%22%5d/cp:conditions/cp:
    identity/
3 cp:one%5b@id=%22sip:shange@146.231.123.109%22%5d
4 ?xmlns(cp=urn:ietf:params:xml:ns:common-policy)
```

Listing 5.2: The URI used to access the *<cp:one id="sip:shange @146.231.123.109"/>* resource from a document

From the *node-selector* in the URI, we can see that we use the element names to traverse to the element of interest. We also use the namespace prefixes to qualify these element names. The association between a namespace and a particular prefix is done in the query part of the URI. Additionally, some symbols in the *node-selector* have to be encoded because they are either not acceptable in HTTP URIs or they are reserved for other purposes. In the URI above; the percentage encoding *%5b* represents "[" and *%5d* represents "]"; *%22* represents double quotes (i.e "). Replacing this in a part of the URI above results in: *cp:rule[@id="closefriends"]* .

In XCAP, we use attribute value comparison to differentiate between multiple elements of the same type. The format *element-id[@attribute-id="unique-value"]* is an XCAP convention for expressing that, amongst all the elements with the name *element-id* we are interested in the one where the value of the attribute, *attribute-id*, is equal to *unique-value*. Therefore, the part of the URI *cp:rule[@id="closefriends"]* expresses that we want to select the rule where the *id* attribute is equal to *closefriends*. The other portion *cp:one[@id="sip:mosioua@146.231.123.109"]* means we are interested in an element named *one* where the value of the *id* attribute is equal to *sip:shange@146.231.123.109*. From this example we can grasp the semantics for XCAP URIs. For instance, if we are interested in

the whole *rule* with the *id* attribute equal to *closefriends*, we use the URI shown in Listing 5.3:

```
1 http://server_address:8080/mobicents/geolocation-policy/users/
    user_id/index/``
2 /cp:ruleset/cp:rule%5b@id=%22closefriends%22%5d
3 ?xmlns(cp=urn:ietf:params:xml:ns:common-policy)
```

Listing 5.3: The URI used to access a *rule* with the *id* attribute of *closefriends*

5.3.2.2 Experimental XML Configuration Access Protocol Element Operations Using the `HttpClient` and the `GeolocationPolicyAppUsage` Classes

In JME, we reused the `GeolocationPolicyAppUsage` class to perform experimental geolocation-policy element operations. We statically put together the URIs discussed in the preceding section to get, add and delete elements. Apart from the values of the *node-selectors*, content types and the content, geolocation-policy element operations are the same as document operations discussed earlier. To add or modify elements, we use HTTP POST requests with content (as an emulation of HTTP PUT requests). We emulated HTTP DELETE requests used for element deletion using HTTP POST with no content. HTTP GET requests are used for getting the elements referenced in the URIs.

From these experiments, we re-organised the client classes into an ad-hoc XCAP SDK. Here, the word *ad-hoc* is used to indicate the fact that the XCAP client makes HTTP DELETE and PUT requests using adapted HTTP POST requests. The functionality of the classes had to be packaged so that they might be exposed to LBSs developers in like fashion to the SIP processes discussed in the previous chapter. Before doing this, however, a decision had to be made regarding the XCAP operations to be supported for the geolocation-policy appusage.

Element	GET	PUT	DELETE
ruleset(document)	X	X	X
rule	X	X	X
conditions	X		
one		X	X
many		X	X
location		X	X
sub-handling		X	
transformations	X		
provide-location		X	X

Table 5.1: The supported geolocation-policy XCAP element operations

5.3.3 Adding XML Configuration Access Protocol Support to the Location-Based Services Development Kit Prototype

5.3.3.1 Required Geolocation-policy XML Configuration Access Protocol Operations

For our ad-hoc XCAP SDK, we chose to support only XCAP operations on the geolocation-policy elements shown in Table 5.1. For reference purposes, Listing 5.1 shows these elements in a sample geolocation-policy document. GET requests are relatively expensive to perform because, processing an XCAP resource from the server requires XML validation and parsing. Therefore, we focused on supporting only the necessary GET operations. In future, more functionality could be added to the JME LBSs SDK.

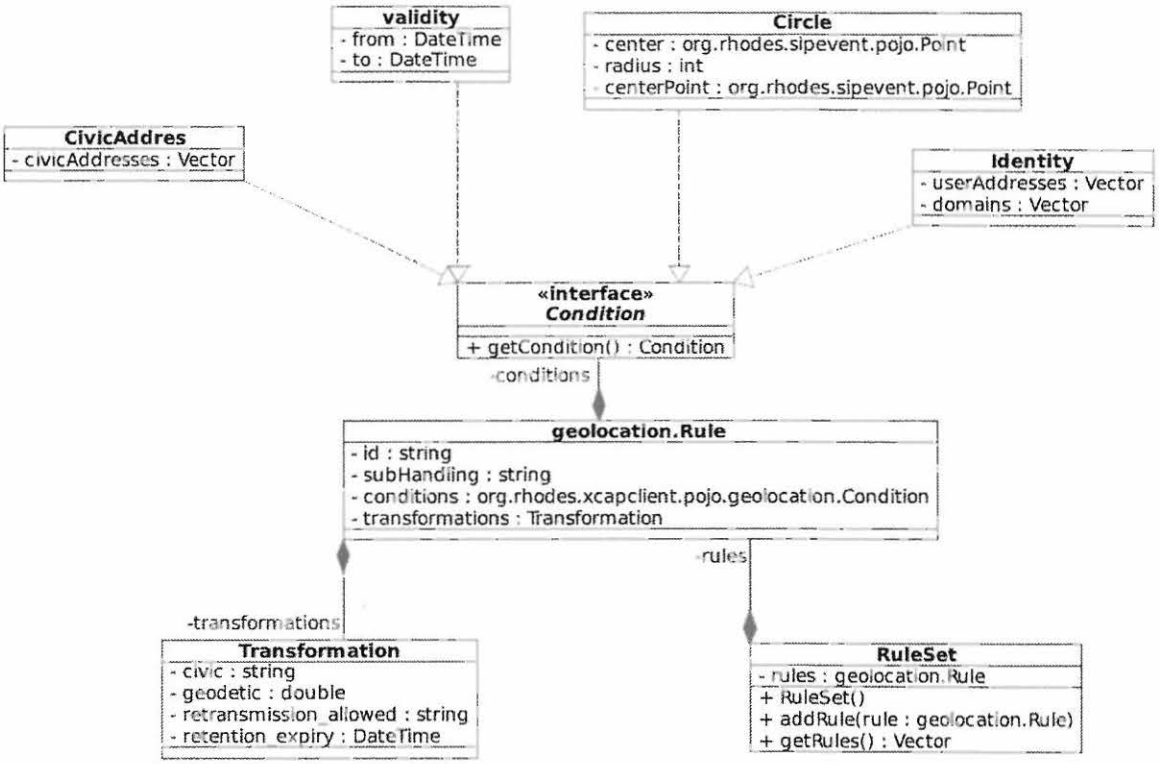

Figure 5.4: Classes for parsing and storing XCAP resources on JME client

5.3.3.2 Parsing and Storing XML Configuration Access Protocol Resources

The ad-hoc XCAP client of the JME LBSs SDK prototype required classes to parse and store the XCAP resources acquired from the XDMS with the GET operations shown in Table 5.1. We put together a class named `RulesetParser` that has static methods to do this. The methods offered invoke the parser at `ruleset` (document), `rule`, `conditions` and `transformations` element levels. The class uses the KXML pull parser discussed in the Chapter 4.

After successfully parsing the resources, the methods return one of the appropriate instances of classes from the `org.rhodes.xcapclient.pojo.geolocation` package that we put together. As can be seen from Figure 5.4, this package contains classes that model the different elements of a geolocation-policy document.

5.3.3.3 The `xcapclient.appusage` Package and the `GeolocationPolicy` Class

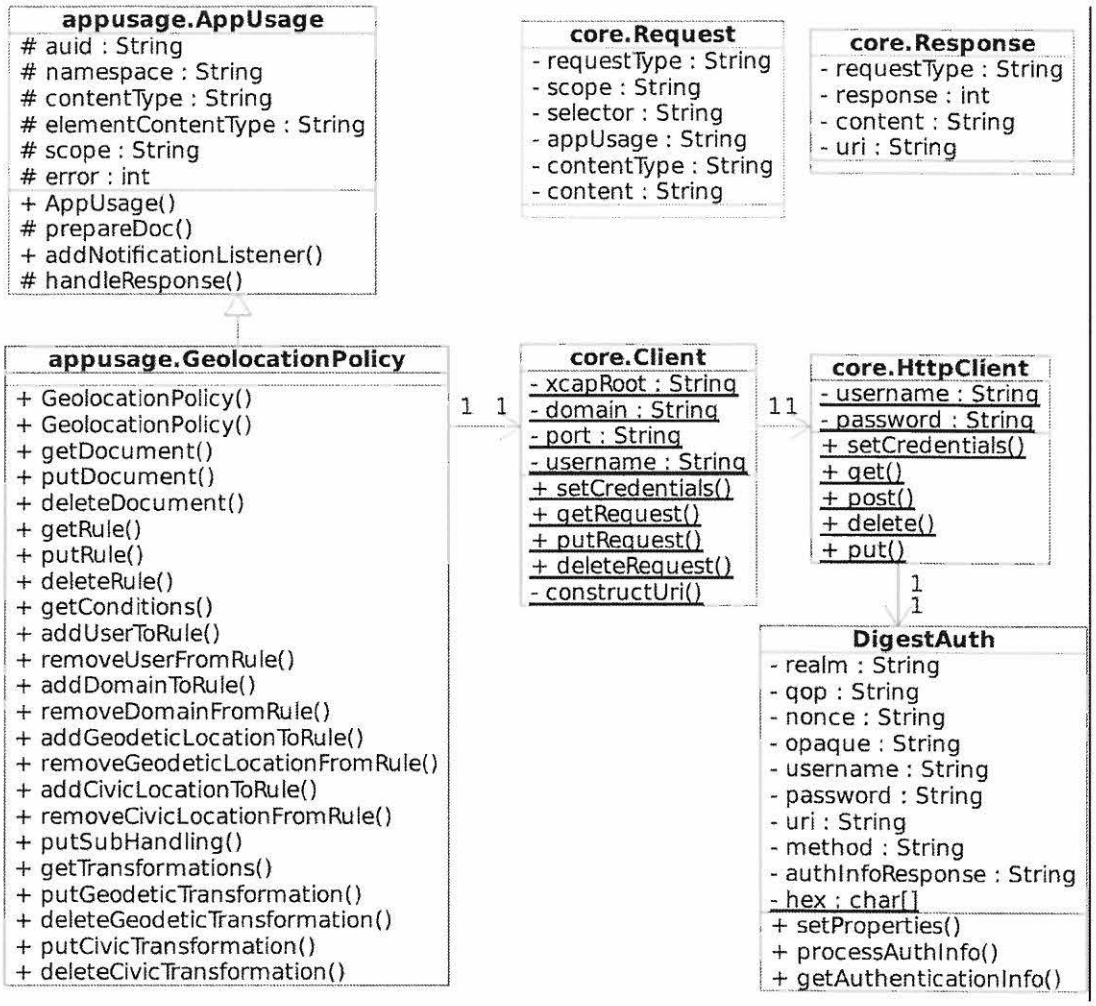

Figure 5.5: The `org.rhodes.xcapclient.appusage` package

Figure 5.5 shows the class diagram of some packages of the ad-hoc XCAP client of the JME LBSs SDK prototype.The `org.rhodes.xcapclient.appusage` package contains classes that handle appusage-specific functions. The abstract `AppUsage` class defines the variables and operations that are common to all appusages. This was done to allow for the easy addition of appusages to the JME LBSs SDK in future. The `GeolocationPolicy` class extends the `AppUsage` class and handles operations that are specific to the geolocation-policy appusage. Such operations include putting together the *node-selector* part of the request

URI and creating and populating the variables of `Request` instances that may be created from the `Request` class of the `org.rhodes.xcapclient.core` package shown in Figure 5.5.

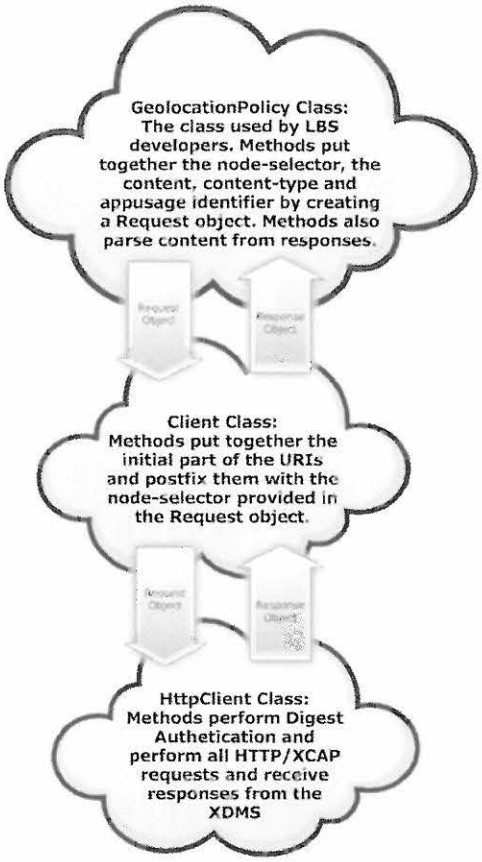

Figure 5.6: The flow of `Request` and `Response` instances in the JME LBSs SDK prototype

The general functionality and flow of events in the ad-hoc XCAP client of the JME LBSs SDK prototype is illustrated in Figure 5.6. To provide deeper insight, we provide a walk-through of the operation of acquiring a `rule` element from the geolocation-policy document stored at the Mobicents XDMS. As can be seen from Figure 5.7, the process begins with a call to the `getRule` method of the `GeolocationPolicy` class and providing it the identifier (`ruleId`) of the `rule`. The method puts together a *node-selector* with the `ruleId` of the `rule` to be fetched from the XDMS. The method passes an instance of the `Request` class and passes it to the `Client` class by invoking the `getRequest` method in step 2.

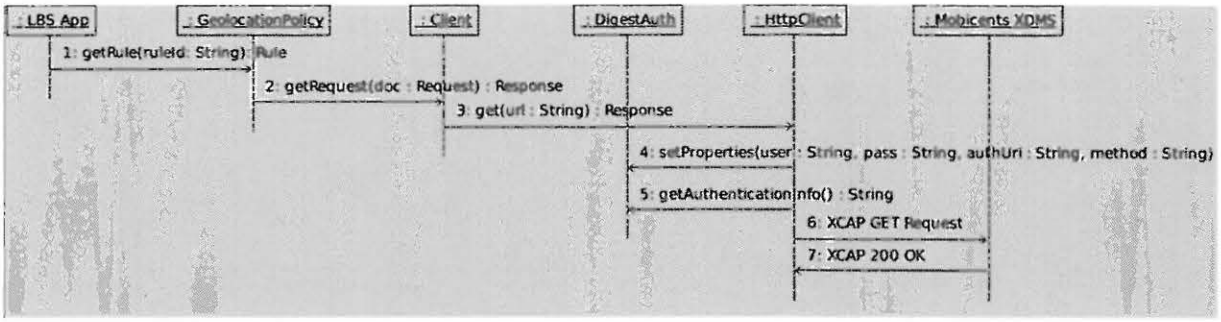

Figure 5.7: A sequence diagram for getting a `rule` from the XDMS

5.3.3.4 The Client Class and the `org.rhodes.xcapclient.core` Package

At step 2 of Figure 5.7, the `Client` class of the `org.rhodes.xcapclient.core` package constructs the initial parts of the XCAP URI using its attributes and those provided in the `Request` instance passed to its `getRequest` method. The constructed URI is then post-fixed with the *node-selector* stored in the `Request` instance to create a complete XCAP URI. The URI is passed to the `get` method of the `HttpClient` class in step 3. Digest authentication is provided by invoking the `setProperties` and `getAuthenticationInfo` methods of the `DigestAuth` class of the `org.rhodes.xcapclient.auth` package shown in steps 4 and 5. Finally, the XCAP get request is sent by the `HttpClient` class to the Mobicents XDMS in step 6. When an XCAP response is received in step 7, an instance of the `Response` class is relayed to the `getRequest` method of the `GeolocationPolicy` class.

This concludes the discussion of the functionality of the JME LBSs SDK prototype XCAP client. For more information, the reader may consult the source code and the Java documentation provided with this thesis.

5.3.3.5 The `GeolocationPolicy` Class As an Interface to the XML Configuration Access Protocol Client of the Location-Based Services Development Kit Prototype

The JME LBSs SDK prototype has been crafted in such a way that an LBSs developer interacts with the `GeolocationPolicy` class only for geolocation-policy appusage functionality. The developer may instantiate a static instance of the class as a part of a Target or

an independent rule-making component. They may then synchronously invoke any of the methods of the class to perform geolocation-policy operations.

5.4 Enabling Privacy-Aware Location Subscription on the Modified Mobicents Presence Service

Looking back at our envisaged privacy architecture shown in Figure 5.1, we can see that we have discussed only aspects related to a Target or rule-making entity getting, creating, modifying and deleting geolocation-policy documents. The subject of this section is the usage of those documents during SIP subscription notifications to filter the PIDF-LO document sent to different LCs.

We discussed the implementation of the SIP subscription process in the previous chapter while leaving out the part regarding authorisation and privacy. By default, if a Target or presentity does not specify any geolocation-policy, the modified MPS authorises subscriptions full PIDF-LO document to all requesting LCs. If a Target has a geolocation-policy document at the XDMS, however, the MPS applies the rules of that policy to the currently published PIDF-LO. This process happens mainly at the `PresenceSubscriptionControlSBB` of the modified MPS. We discuss the process of implementing this privacy-aware SIP location subscription on the modified MPS in the subsections that remain.

5.4.1 Phase One: Authorising Subscriptions with the Identity Condition

We have already discussed the states shown in Figure 5.8 for Targets that do not have privacy policies in Chapter 4. Here, we are going to discuss the privacy-related aspects. In terms of privacy, the first process invoked at the `PresenceSubscriptionControlSBB` of the modified MPS, when an initial subscription request is made by an LC, is subscription authorisation.

First the reader needs to understand that the geolocation-policy documents of the Targets are accessed from the Mobicents XDMS using the `XDMClientControlSBB` discussed in Chapter 2. All the geolocation-policy documents of the Targets currently subscribed to are stored in cache by the `PresenceSubscriptionControlSBB`. The instances of the JAXB

bindings we created in section 4.5.2.1 of Chapter 4 and section 5.2.1 of this chapter are utilised here to parse the content of the published PIDF-LO and the geolocation-policy documents.

The process of subscription authorisation entails checking if the address or domain of the LC is present in the `identity` element of any of the rules in the geolocation-policy document put on the XDMS by the Target. From Figure 5.8, it can be seen that the subscription-state is set to `pending` from the first subscription request until authorisation is provided. If the address or domain of the LC is present in the `identity` element of the geolocation-policy document, then the LC is authorised and the subscription-state changes to `active`. Originally, the MPS had functionality to process identity conditions for presence policies. The modifications we made to it to enable subscription authorisation entailed the loading of geolocation-policy schemas and making reference to the geolocation-policy documents of the geolocation-policy appusage.

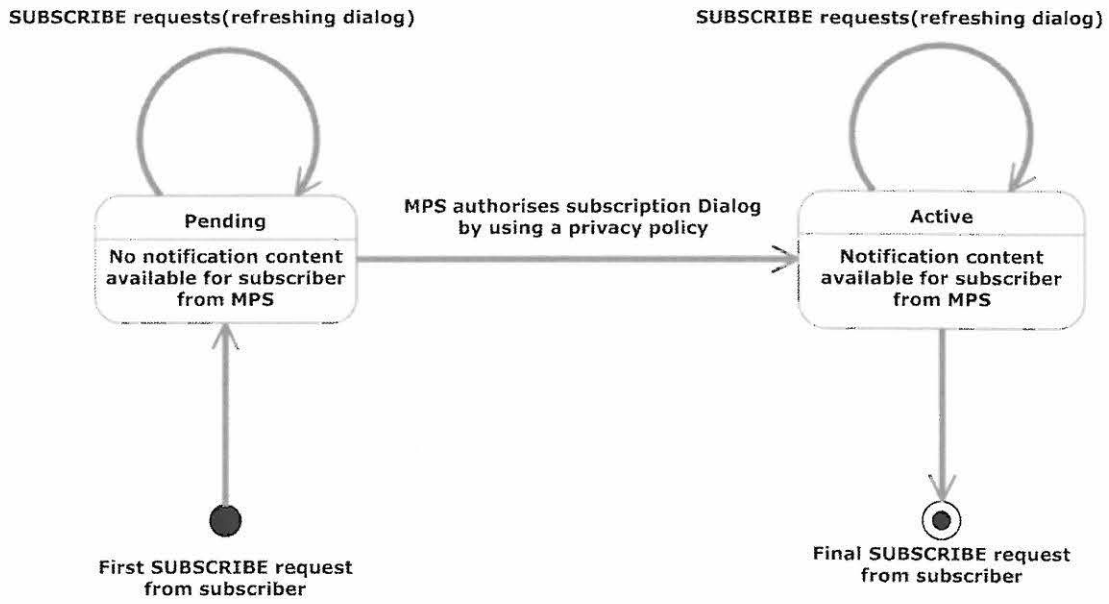

Figure 5.8: The subscription message sequence (with authorisation)

An authorised subscription yields content in the notification requests made to the LC by the XDMS. Before this can happen, the rest of the geolocation-policy rule has to be applied. Otherwise, if the address or domain of the subscriber cannot be found, then the subscriber is unauthorised and the subscription-state remains `pending`. Unauthorised subscriptions

yield no content in the notification requests as can be seen from Figure 5.8. We mentioned in the previous chapter, while implementing a subscription mechanism on the client, that we maintain subscription dialogs with a `pending` status for a few subscription refresh requests only. Thereafter, the SIP LC must stop the subscription dialog.

5.4.2 Phase Two: Evaluating the Location and Validity Conditions

Like identity condition processing, validity condition evaluation was already implemented as part of the MPS. For location conditions, we modified the `PresenceSubscriptionControlSBB` class such that it uses its helper class, `RulesetProcessor`, for determining if the currently published location information (in the PIDF-LO) is contained within any one of the `location` elements (belonging to the geolocation-policy document of the Target). This is done within the `processConditions` method of the `RulesetProcessor` class.

We mentioned in the previous chapter that we limit the publication of geodetic information to points, circles and polygons only. However, for privacy-aware location subscription, only privacy for geodetic points published in the PIDF-LO documents is implemented. The geodetic location condition calculations are based on whether the currently published point (pair of latitude and longitude coordinates) is located within any of the Geoshape circles expressed in the geolocation-policy document (see section 5.1.2). The method we employed uses the *haversine* formula, which is used to determine the shortest distance between two points over the earth's surface [80]. We deduce whether a point is within a circle by calculating if the distance between that point and the center point of the circle is less than the radius of the circle. The `processConditions` method calls on the `inCircle` method of the `CircleConditon` class that we put together for this operation.

We also mentioned in the previous chapter that civic information publications are limited to only five levels of the seventeen levels that can be expressed. Civic location condition evaluation is simpler, as it involves basic string comparison between the different civic information levels. For instance, if one of the `location` elements of the `location-condition` element states that the Target should have their `country` element set to *DE* then a PIDF-LO with the `country` element set to *DE*, will result in that `location` element evaluating to true. If the Target has specified a validity condition, then the `RulesetProcessor` class checks that condition against the current time as well. This concludes the discussion on processing conditions for geolocation-policy documents.

5.4.3 Phase Three: Applying Transformations

The discussion given in this section is partly taken from [40]. The child elements of the `conditions` element (i.e `identity`, `location` and `validity`) must evaluate to true for the `transformations` element of the rule to be evaluated [66]. We mentioned this fact in section 5.1.2. The contents of the `transformations` element change the currently published Location Object (from the PIDF-LO) to the appropriate values specified by the Target for the LCs in the `identity` element of the rule. In other words PIDF-LO filtering parameters are expressed in the `transformations` element of the rules.

5.4.3.1 Implementing Geodetic Transformations

A geodetic transformation in the `provide-geo` element, as shown in Figure 5.1, can be used to reduce the granularity of the geodetic information published by the Target in its Location Object. A geodetic transformation may be expressed in the geolocation-policy document of a Target as an integer representing a radius [66].

Figure 5.9 illustrates how a geodetic transformation is used during SIP subscriptions to protect the privacy of a Target. A Target publishes its location, point *(lat, long)*, in a PIDF-LO document to the modified MPS. Assuming that LCs A and B are part of the same rule in the geolocation-policy document of the Target, the modified MPS uses point *(lat, long)* together with radius, *r1*, provided in the rule of the policy, to generate new random locations (geodetic points). LC A receives the generated point *(lat1, long1)* as the location of the Target and LC B the generated point *(lat2, long2)*. Both these points are within a circle which has the center point *(lat, long)* and *r1* as its radius.

Another LC (C) in a different rule of the geolocation-policy document might be subscribed to the Target's location. The transformational radius of the rule is *r2*. LC B, gets a different point *(lat3, long3)* generated in the same way as *(lat1, long1)* but using *r2* as the transformational radius. The Target can vary the accuracy of the information shared with different LCs by changing the transformational radius of different rules. This process of reducing the resolution of the location information happens during every notification that bears content. This privacy mechanism was implemented as specified in [66].

We implemented the functionality of applying transformations to Location Objects in the `filterContentPerSubscriber` method of the `PresenceSubscriptionControlSBB`. The process of applying geodetic transformations is achieved by using a basic trigonometric

formula that is used to determine coordinates of points (on the earth's surface) on the circumference of a circle given its center point, radius and an angle. However, our requirement was to generate a random point that is within the circle and not on the circumference. To do this, we generate a random radius ($r1$) between zero and the real radius of the circle and also a random angle between zero and 360 degrees. The the `filterContentPerSubscriber` method uses the `ambiguateCircle` method of the `CircleConditon` class we created to achieve this.

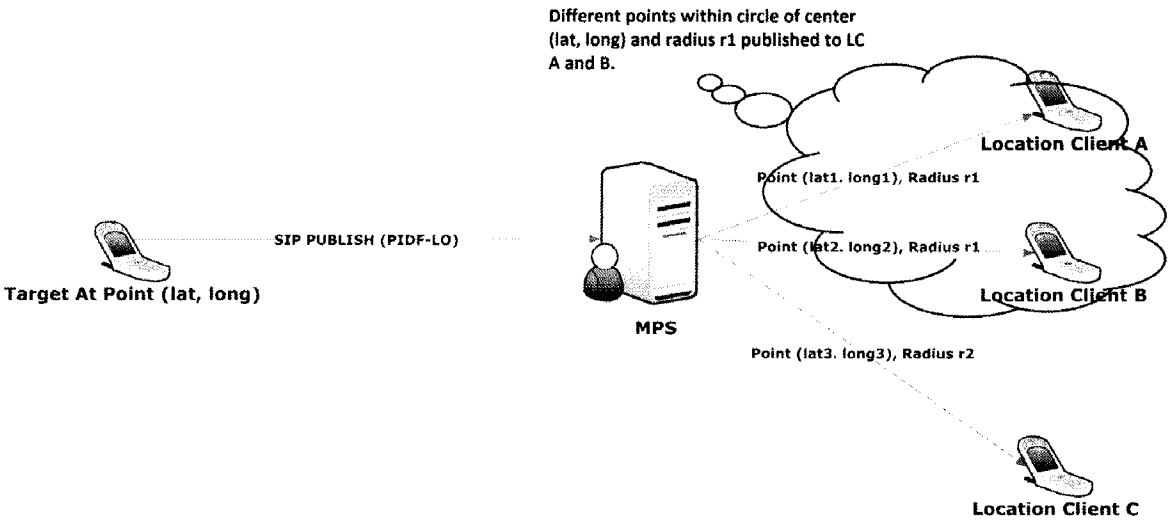

Figure 5.9: Applying geodetic transformations, adapted from [66]

5.4.3.2 Implementing Civic Transformations

A Target or an independent Rule Maker can specify a civic transformation that can reduce the granularity of the civic location information it has published in its Location Object. We have mentioned in the last chapter and in the preceding sections that we have chosen

to support only five of the many civic location fields in our implementation. Civic transformations can be expressed by selecting either one of the six values for the `provide-civic` element. These values are: `full`, `building`, `city`, `region`, `country` and `none` [66].

We implemented the civic transformations in the `filterContentPerSubscriber` method of the `PresenceSubscriptionControlSBB`. Transformational values of either `full` or `building` result in an unaltered Location Object. This is because our civic location resolution does not go as deep as `building` level. A transformational value of `city` leads only to the `country`, `A1` and `A2` elements of the Location Object's civic information being provided to the LC. The Location Object's civic information is altered in this way according to the policy specified by the Target. We do not discuss the `region`, `country` transformational values but we have implemented them in similar fashion. A transformational value of `none` results in no civic information being provided to the LC at all.

5.5 Chapter Summary

In this chapter, we have discussed the implementation of the geolocation-policies in both the JME LBSs SDK prototype and the modified MPs. We have provided a discussion on the privacy architecture that we aimed to implement for our project. We briefly introduced the geolocation-policy document, as it is crucial to the realisation of the privacy functionality. The first step of the implementation involved the addition of the geolocation-policy application usage (appusage) to the Mobicents XDMS. Such a step was required so that Targets or separate rule-making entities would be able to add, modify, get and delete geolocation-policy documents at the XDMS. The second step involved the creation of the XCAP client to the JME LBSs SDK prototype. The client is capable of performing operations on the geolocation-policy appusage at the XDMS. The XCAP client was first implemented by experimenting with basic XCAP requests to provide an understanding of the requirements. Thereafter, the XCAP SDK was put together such that LBSs developers could easily invoke client geolocation-policy operations to enable LBSs privacy using the JME LBSs SDK prototype. The final step of the privacy implementation involved the further modification of the MPS so that it uses the geolocation-policy documents specified by the Targets during SIP notifications.

Chapter 6

Consolidating the Location-Based Services Tools and Demonstrating Their Use

In Chapters 4 and 5, we discussed the development of the JME LBSs SDK prototype and modified the MPS. These steps involved setting up facilities to cover the transport and privacy protection requirements of SIP-based LBSs. The main aim of this chapter is to show how we consolidated and improved these tools and exercised their functionality.

Firstly, we improved the developer-friendliness of the toolkit and present the `org.rhodes.LBSs` package that we have put together. The package contains an abstract class and interfaces that can be used to aid developers in creating LBSs. Secondly, we put together an application to support the general location-based requirements of a friend-finder service and another for a child-tracker service. In short, a friend-finder service is a social networking service that allows friends to locate each other in like fashion to Google Latitude. A child-tracker service is one that allows parents to be notified when their children leave designated areas, such as playgrounds or parks. One of the major reasons why we chose to implement these two services is that their requirements allow us to add new functionality to the JME LBSs SDK prototype and to the modified MPS. Our work is on the core functionality (transport and privacy protection) of cross-referencing LBSs and not on their presentation. For this reason, we ignore the Graphical User Interfaces (GUIs) of the applications completely.

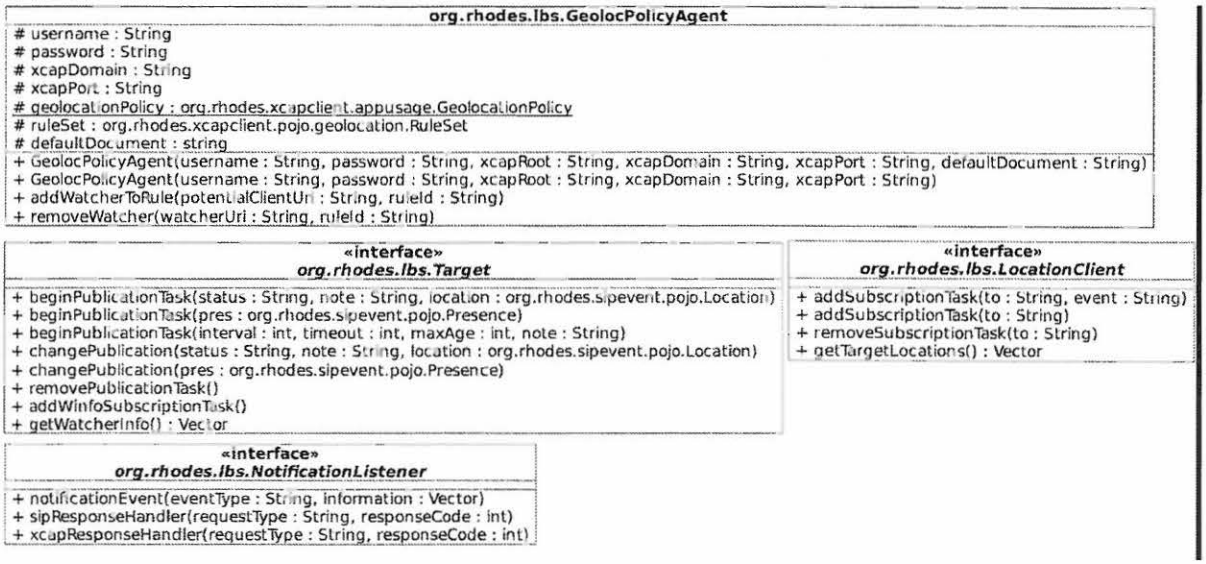

Figure 6.1: The interfaces and classes of the `org.rhodes.LBSs` package

6.1 Improving the Location-Based Services Development Kit Prototype

The JME LBSs SDK prototype discussed thus far is designed to support the development of LBSs through the use of the `SipConnectionManager` and the `GeolocationPolicy` classes. We noted that there were two major shortcomings in our SDK prototype. The first was the fact that responses received from the MPS and XDMS are visible only to objects within the SDK itself. They are not made available to service developers so that they could use them in their applications. This problem is addressed in section 6.1.1.

The second shortcoming is that there are no interfaces and abstract classes to aid developers in putting together common LBSs entities such as Targets and LCs. This problem is addressed in section 6.1.2. The interfaces and abstract classes discussed in this section can be found in the `org.rhodes.LBSs` package shown in Figure 6.1. The reader or LBSs developer may consult both the Javadoc and the source code provided with this thesis for further reference.

6.1.1 Listening to Server Requests and Responses in Location-Based Services Applications

When SIP or XCAP responses are received from the MPS and XDMS, respectively, objects in the JME LBSs SDK prototype process them. In the case of SIP responses, such processing is limited to maintaining SIP connections and dialogs. LBSs developers will most likely need to perform operations based on these responses or to present them to the users of their services. Additionally, requests such as SIP notifications and responses to XCAP `GET` requests may bear content that will be used in the LBSs applications developed with the toolkit. To make the server requests and responses accessible to LBSs developers, we created the NotificationListener interface in the org.rhodes.LBSs package. Entities implementing this interface indicate their interest in SIP and XCAP responses by invoking the `addNotificationListener` methods (discussed in 6.1.1.2) of the `SipConnectionManager` and the `GeolocationPolicy` classes, respectively.

6.1.1.1 The `NotificationListener` Interface

As illustrated in Figure 6.1, the `NotificationListener` interface contains three methods, namely `notificationEvent`, `sipResponseHandler` and `xcapResponseHandler`. The `notificationEvent` method is notified of presence and location information by the `SipConnectionManager` class when SIP notification requests are received from the modified MPS. By implementing the `notificationEvent` method, LBSs applications do not need to poll continually for location information using the `getSubscriptionContent` method of the `SipConnectionManager` class (discussed in Chapter 4).

The `sipResponseHandler` method of the `NotificationListener` interface is notified by the `SipConnectionManager` class when SIP responses are sent to the mobile device from the modified MPS. LBSs developers can handle these responses in their applications by implementing this method. The method is provided with a string indicating the request method and the response code as can be seen in Figure 6.1.

Finally, the `xcapResponseHandler` method of the `NotificationListener` interface provides to LBSs developers means to be notified by the `GeolcationPolicy` class when XCAP responses are made to the mobile application by the XDMS. Like the `sipResponseHandler` method, the `xcapResponseHandler` method provides information about the request method that generated the response and the associated response code.

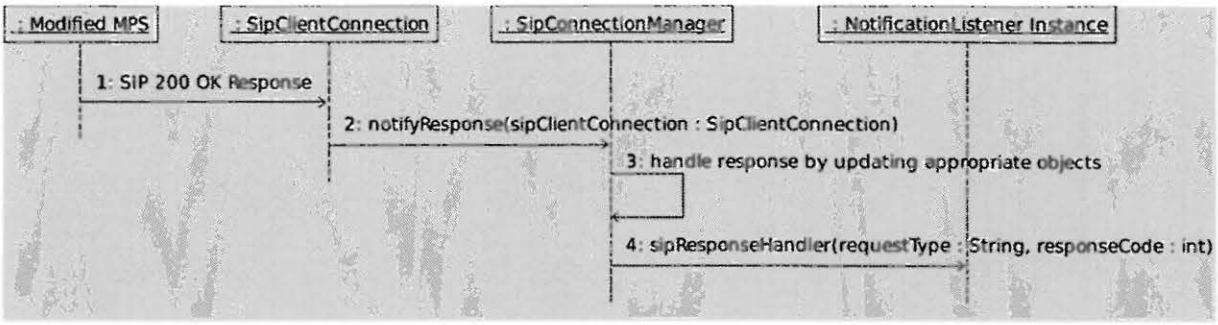

Figure 6.2: A sequence diagram showing the use of the `NotificationListener` interface

For SIP and XCAP responses to be delivered to `NotificationListener` instances from the `SipConnectionManager` and `GeolcationPolicy` classes, respectively, `addNotificationListener` methods were added to both classes.

6.1.1.2 The `addNotificationListener` Methods

Instances of the `NotificationListener` interface need to register their interest in SIP and XCAP events by invoking the `addNotificationListener` methods of the `SipConnectionManager` and `GeolcationPolicy` classes, respectively. This enables SIP and XCAP responses and requests to be sent to the `NotificationListener` instances as they are received from the modified MPS and XDMS. Figure 6.2 shows an example of a SIP 200 OK response that is sent by the modified MPS to a mobile application. The response is received and processed within the `notifyResponse` method of the `SipConnectionManager` class discussed in Chapter 4. It is then sent to the `handleResponse` method of the same class for internal processing. Thereafter, the `handleResponse` method invokes the `sipResponseHandler` method of the `NotificationListener` instance added using the `addNotificationListener` method. The `notificationEvent` method of the `NotificationListener` instance is invoked in similar fashion when SIP notifications that have content are received.

Adding `NotificationListener` instances to the `GeolocationPolicy` class using its version of the `addNotificationListener` method allows for its methods to invoke the `xcapResponseHandler` methods of these instances. This way, XCAP responses are made available to `NotificationListener` instances as soon as they are received from the XDMS.

6.1.2 Interfaces and Abstract Classes for Guiding the Creation of Location-Based Services

6.1.2.1 Guiding the Creation of Target Applications

For the easy creation of Target entities, LBSs developers must implement the `Target` interface shown in Figure 6.1. As it is stated in our Javadoc, there must be only one Target per mobile device at a given time. Figure 6.1 shows that the `Target` interface has three overloaded `beginPublicationTask` methods; two `changePublication` methods; and a `removePublicationTask` method.

The overloaded `beginPublicationTask` methods provide developers with three alternative ways of beginning presence and location information publications. The first version of the method requires a presence status, note and a `Location` instance (from the `org.rhodes.sipevent.pojo` package) as parameters. The second version of the method requires a `Presence` instance only. The final version begins location publications using the mobile device's location provider such as GPS. Target applications could begin publications through one of these methods depending on the requirements of the application being created.

The `changePublication` methods guide the developers in changing the presence and location information being published to the modified MPS by the application. The first version of the method requires a presence status, note and an instance of the `Location` class as parameters. The second version requires an instance of the `Presence` class only. The `removePublicationTask` method guides the implementation of the process of halting presence and location publications made by the application.

6.1.2.2 Guiding the Creation of Location Client Applications

Another interface we created in the `org.rhodes.LBSs` package is the `LocationClient` interface. This interface can be used to guide the creation of LC applications. As it is stated, there must be only one LC entity per mobile device. This LC entity can watch the presence and location of many Targets from the modified MPS. As it is illustrated in Figure 6.1, the `LocationClient` interface contains three methods: the `addSubscriptionTask` method; the `removeSubscriptionTask` method and the `getTargetLocations` method.

The `addSubscriptionTask` method creates a new SIP subscription dialog to a particular Target for a `LocationClient` instance. It takes in a parameter bearing the address of

the Target to be subscribed to. The `removeSubscriptionTask` method removes a SIP subscription dialog belonging to the Target provided as a parameter for a particular `LocationClient` instance. Finally, the `getTargetLocations` method is used for polling for notification content on all active subscription dialogs made by the LC. The method returns a collection of instances of the `Presence` class (from org.rhodes.sipevent.pojo package). As discussed in section 6.1.1.1, the polling mechanism may not be suitable for many LBSs, as it could be wasteful. Therefore, most LC applications are expected to implement the `NotificationListener` interface to acquire location and presence event notifications from the `SipConnectionManager` class instead of implementing the `getTargetLocations` method.

6.1.2.3 Guiding the Creation of Applications that Manipulate Geolocation-Policy Documents

At this point, we have discussed the interfaces, `Target` and `LocationClient`. The methods provided in these interfaces can be implemented to create both Target and LC entities for LBSs applications. In this section, we discuss the `GeolocationPolicyAgent` abstract class in the `org.rhodes.LBSs` package. This class can be extended to aid developers in designing and implementing applications for creating, getting, modifying or deleting geolocation-policy documents at the XDMS. The `GeolocationPolicyAgent` abstract class should normally be extended by instances of the Target interface or by other rule-making entities. Figure 6.1 shows that the class has attributes for holding values necessary for making authenticated XCAP requests such as a `username` and a `password`. Other attributes such as the `xcapDomain`, `xcapPort` and `xcapRoot` are necessary for constructing XCAP URIs and for making requests to the XDMS. The constructor of the `GeolocationPolicyAgent` class creates an instance of the `GeolocationPolicy` class for the developer. Connections to the XDMS and operations on the geolocation-policy document are made by invoking methods of the `GeolocationPolicy` instance using the attributes of the `GeolocationPolicyAgent` abstract class.

The constructor of the `GeolocationPolicyAgent` class performs activities that are necessary for all entities that manage geolocation-policy documents at the XDMS. These activities include, first, attempting to get the geolocation-policy document of the Target from the XDMS. If acquired successfully, the geolocation-policy document is parsed and stored locally. In some situations, the user might not have created a profile for the Target on the XDMS or the username or password combinations used are wrong. This scenario

results in a `GeolocationPolicyAgent` instance not being authorised to perform operations at the XDMS. The `GeolocationPolicyAgent` instance detects this and indicates that a user has to create a profile on the XDMS using the web interface to solve this problem. In first-time-access situations, a Target will not have a geolocation-policy document stored at the XDMS. The `GeolocationPolicyAgent` instance puts a default geolocation-policy document for the Target in this situations. Other XCAP errors could be accessed by implementing the *xcapResponseHandler* method of the `NotificationListener` interface as discussed in 6.1.1.1.

Having discussed these improvements to the JME LBSs SDK prototype, we move on to designing and implementing the friend-finder and child-tracker services. These services are meant to serve as demonstrations of how LBSs tools could be used, but also add extra functionality to them.

6.2 The Friend-Finder Service

6.2.1 Application Requirements

A friend-finder service is a service where users see the location of their friends, as discussed in [40]. The service requires the addition, removal and viewing of friends' locations and presence information. Each user should have an application with capabilities of both Target and LC entities since users need to publish and subscribe to share location and presence information. The use cases for the application are as shown in Figure 6.3.

6.2.1.1 Use Cases at The Target

Firstly, Target instances need to make registrations to the SIP proxy and location service so that their current contact details are stored there. Secondly, the Target requires the means to publish their location information. Thirdly, the Target needs to be able to create, update and delete their geolocation-policy documents at the XDMS. Finally, a Target should be able to add, remove and view friends from its geolocation-policy document. These are the only use cases that we want to address as part of the friend-finder Target application.

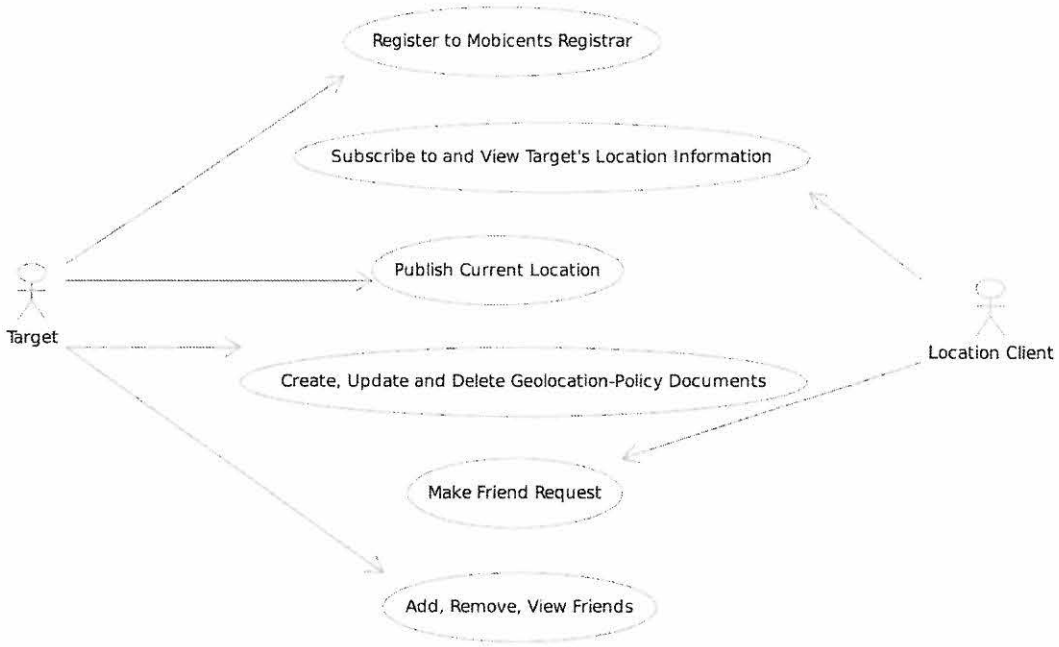

Figure 6.3: The friend-finder application use cases

6.2.1.2 Use Cases at The Location Client

The LC part of the friend finder application is responsible for fulfilling two use cases: subscribing Targets' locations and viewing this information, and making friend requests. Subscribing to friends involves the process of initiating and maintaining subscriptions to their real-time location and presence information. Making friend requests allows the LC to indicate to potential friends a wish to receive location information from them.

6.2.2 Implementing the Service Using the Location-Based Services Development Kit Prototype

6.2.2.1 Creating the `FriendFinder` Class

Figure 6.4 shows that the interfaces and classes provided in the `org.rhodes.LBSs` package are used in the implementation of the friend-finder service. The `FriendFinder` class, which is the only class of the service, implements both the `Target` and `LocationClient` interfaces. This is required because each friend-finder application needs both Target and

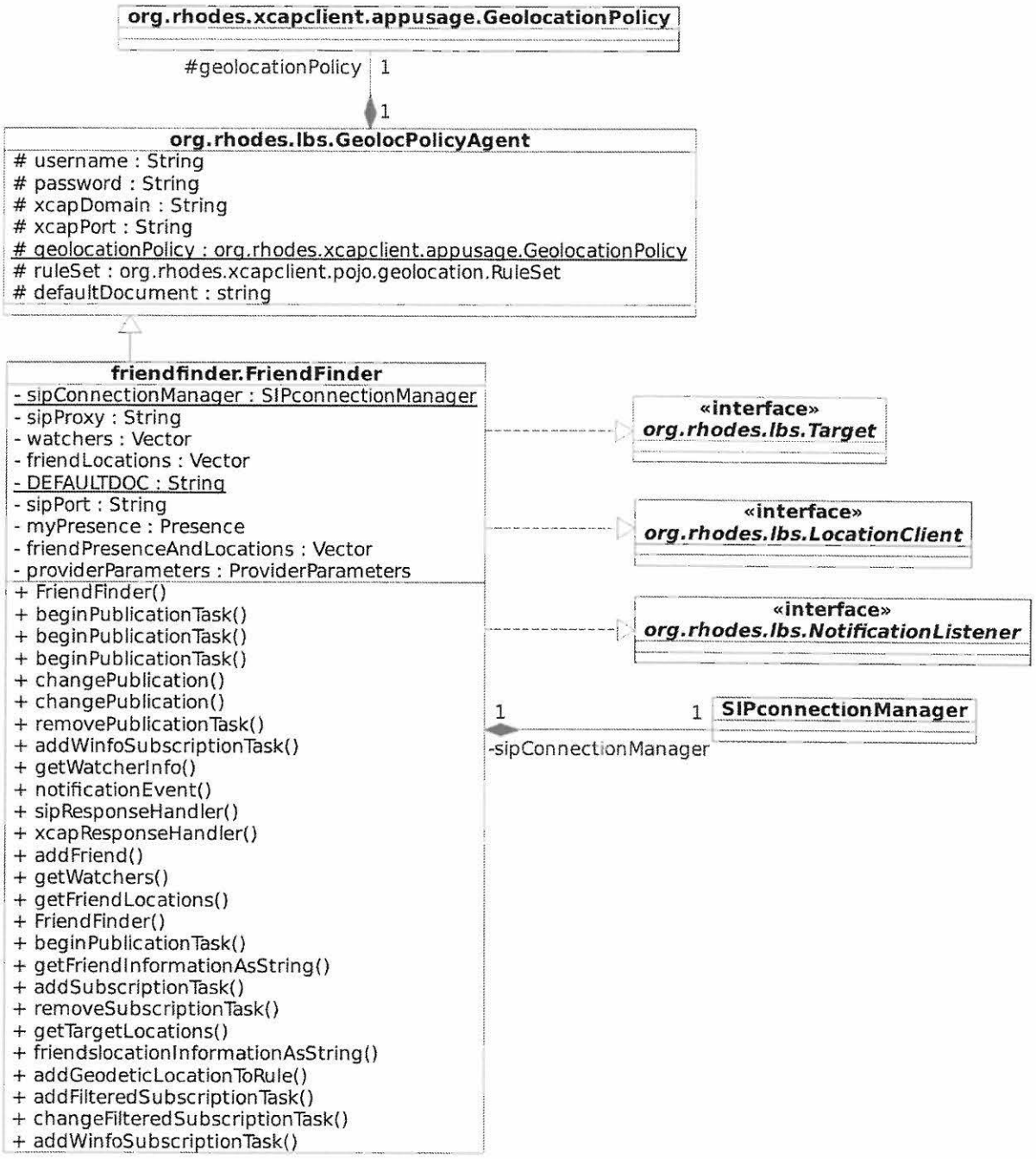

Figure 6.4: The main classes used in the friend-finder application

LC functionality as we mentioned earlier in the service requirements. The class implements the `NotificationListener` interface to enable it to receive SIP and XCAP request and response event notifications from the `SipConnectionManager` and `GeolocationPolicy` instances. The class extends the `GeolocationPolicyAgent` abstract class so that it may easily perform geolocation-policy document operations.

6.2.2.2 Invoking Initial Geolocation-Policy Document Processing and Starting Session Initiation Protocol Registrations

The `FriendFinder` class constructor takes in several parameters to allow it to initialise both its `SipConnectionManager` and `GeolocationPolicy` instances. This is as shown in Listing 6.1. The `GeolocationPolicy` instance is created through invoking the constructor of the extended `GeolocationPolicyAgent` abstract class in line 2. By calling this parent constructor, the initial geolocation-policy document processing (discussed in 6.1.2.3) is performed as well.

The `SipConnectionManager` instance is created in the `FriendFinder` class in lines 5 and 6. The instance automatically initiates the SIP registration process upon creation as discussed in Chapter 4. The *Register to Proxy* use case is easily implemented by creating this instance. The `SipConnectionManager` constructor takes in two parameters for SIP ports, one for the server and one for the local mobile device. In lines 8 and 9, the `FriendFinder` class adds itself as the `NotificationListener` to both its `SipConnectionManager` and `GeolocationPolicy` instances. By doing this, the class becomes a listener to SIP and XCAP requests and responses.

```
1 public FriendFinder(String username, String password, String
   xcapRoot, String xcapDomain, String xcapPort, String sipProxy,
   String sipPort)
2 {   super(username, password, xcapRoot, xcapDomain, xcapPort,
   DEFAULTDOC);
3     this.sipProxy = sipProxy;
4     this.sipPort = sipPort;
5     sipConnectionManager =
6     new SIPconnectionManager(username, sipProxy, sipPort, sipPort);
7     sipConnectionManager.addNotificationListener(this);
8     geolocationPolicy.addNotificationListener(this);
9
```

```
10 ...
11 }
```

Listing 6.1: A portion of the `FriendFinder` class constructor

6.2.2.3 Publishing the Current Location of the Target

We have mentioned in 6.2.2.1 that the `FriendFinder` class implements the `Target` interface's methods for location publication. For the friend-finder service, we chose to implement two alternative methods for initiating publications and only one method for changing them. One of the methods begins the publication of location and presence from the provided presence `status`, `note` and an instance of the `Location` class. The implementation of this method is as shown from the beginning of line 2 to line 11 in Listing 6.2. First we create a new `Presence` instance to keep track of the presence and location information to be published (in lines 5 to 8) in the `FriendFinder` class. We then publish the information by invoking the `beginPublicationTask` method of the `SipConnectionManager` instance.

```
1  // Abstratct methods from the Target interface.
2  public void beginPublicationTask(String status, String note,
     Location location)
3  {
4     //Create new Presence object to keep track of the presence and
        location information.
5     myPresence = new Presence();
6     myPresence.setStatus(status);
7     myPresence.setNote(note);
8     myPresence.setLocation(location);
9     //Publish the presence and location information.
10    sipConnectionManager.beginPublicationTask(status, note, location)
        ;
11 }
12
13 // publish by using GPS or other available location provider
14 public void beginPublicationTask(int interval, int timeout, int
     maxAge, int accuracy, String status, String note)
15 {
16    // store parameters for location provider
```

```
17      providerParameters =
18      new ProviderParameters(interval, timeout, maxAge, accuracy);
19      myPresence = new Presence();
20      myPresence.setStatus(status);
21      myPresence.setNote(note);
22      //begin publishing
23      sipConnectionManager.beginPublicationTask(interval, timeout,
          maxAge, accuracy, status, note);
24      //get the location published by the provider and store it
          locally
25      myPresence.setLocation(sipConnectionManager.getMyPresence().
          getLocation());
26  }
```

Listing 6.2: Alternative methods for initiating publications in the `FriendFinder` class

An alternative method to start location and presence publications uses GPS or other available location providers on the mobile device. Its implementation is as shown from line 14 to line 25 of Listing 6.2. We provide the method with parameters that are used in setting up the location provider and with basic presence information. The `ProviderParameters` instance created in lines 17 and 18 is used to store the location provider parameters in the `FriendFinder` class. In lines 19 to 21, we create a `Presence` instance to store the presence and location information locally.

To modify the currently published information of the `FriendFinder` class, we implement one of the `changePublication` methods of the `Target` interface. The version of the method we chose to implement modifies the information published by replacing it with the presence `status`, `note` and a `Location` instance that are provided as its parameters. The functioning of this method is similar to that of the `beginPublicationTask` method except that it invokes the `changePublication` method of the `SipConnectionManager` instance to refresh the location and presence information being published with the new one. The process of removing or stopping the publication process is performed by implementing the `removePublicationTask` method.

Implementing the methods of the `Target` interface as discussed above addresses the requirements of the *Publish Current Location* use case. SIP responses generated from the invocation of the publication methods can be accessed within the `notificationEvent` method of the `FriendFinder` class. Since the `SipConnectionManager` instance already

maintains SIP tasks. these responses will most likely be used only for presentation and reporting purposes.

6.2.2.4 Subscribing to the Location and Presence of Friends

Subscriptions to the location and presence information of friends is guided by the implementation of the `LocationClient` interface in the `FriendFinder` class. The `addSubscriptionTask` method of the class is used to invoke the `addSubscriptionTask` method of the `SipConnectionManager` instance. Both these methods take in the SIP address of the Target or resource being subscribed to. The `SipConnectionManager` instance keeps track of all the active subscription dialogs. The `FriendFinder` class can poll for this information by implementing the `getTargetLocations` method as shown in lines 1 to 5 of Listing 6.3. Lines 7 to 34 of the Listing show the `friendsLocationInformationAsString` method of the `FriendFinder` class. This method converts the location and presence information of the Targets (friends) to a string that can be presented to a user.

```
1  public Vector getTargetLocations()
2  {
3   friendPresenceAndLocations = sipConnectionManager.
      getTargetLocations();
4   return friendPresenceAndLocations;
5  }
6
7  public String friendsLocationInformationAsString()
8  {
9   getTargetLocations();
10  Enumeration friends = friendPresenceAndLocations.elements();
11  String locationsAsString = "";
12  while (friends.hasMoreElements())
13  {
14    Presence pres = (Presence) friends.nextElement();
15   locationsAsString = "Address: " + pres.getSipAddress() + " Status:
       " + pres.getStatus() + " Note: " + pres.getNote();
16   if (pres.getLocation() instanceof Point)
17   {
18      Point p = (Point) pres.getLocation();
```

```
19          locationsAsString = " Latitude: " + p.getLatitude() + "
              Longitude: " + p.getLongitude() + "\n";
20      }
21      if (pres.getLocation() instanceof CivicAddress)
22      {
23          CivicAddress civicAddress = (CivicAddress) pres.getLocation();
24          locationsAsString = "Civic location information: \n";
25          Enumeration cv = civicAddress.getCivicAddress().elements();
26          while (cv.hasMoreElements())
27          {
28            String civic = (String) cv.nextElement();
29            locationsAsString = civic + " \n";
30          }
31       }
32      }
33     return locationsAsString;
34  }
```

Listing 6.3: The `getTargetLocations` and `friendslocationInformationAsString` methods of the `FriendFinder` class

The `FriendFinder` Class can also remove subscription dialogs to friends in whom the user is no longer interested by implementing the `removeSubscriptionTask` method of the `LocationClient` interface. This method takes in the SIP address of the Target involved and invokes the `removeSubscriptionTask` method of the `SipConnectionManager` class.

By implementing the methods of the `LocationClient` interface discussed in this section, we fulfill the general requirements of the *Subscribe to and View Friends' Locations and Presence Information* use cases.

6.2.2.5 Specifying and Modifying Default Geolocation-policy Document Rules

The *Specify or Modify Location Privacy Policy* use case is implemented by using the different methods provided by the `GeolocationPolicy` instance. This instance is created in the `FriendFinder` class by extending the `GeolocationPolicyAgent` abstract class. The constructor of the `GeolocationPolicyAgent` can be provided a default geolocation-policy document that a user can customise to suit their privacy needs.

To keep the friend-finder service implementation simple and clear, we limited ourselves to using two rules namely `closeties` and `looseties` in the default geolocation-policy document. These rules differ only in what they provide within their `transformations` elements. The `closeties` rule provides no privacy filtering and the location information is provided to the LCs as it is. The `looseties` rule provides location-based privacy filtering with a transformational radius of five kilometers. If the publication contains civic information it is reduced to `city` level (i.e level of the `A2` civic address element).

```
1 public void addGeodeticLocationToRule(String rule, String placeName,
    Circle circle)
2 {try {
3       geolocationPolicy.addGeodeticLocationToRule(rule, placeName,
          circle);
4       if(geolocationPolicy.getResponse()== HttpConnection.HTTP_OK)
5       {
6        System.out.println("[FriendFinder]Rule: " + rule+ " for place
          : "
7        + placeName + " added successfuly.");
8       }
9   ...
```

Listing 6.4: The `addGeodeticLocationToRule` method of the `FriendFinder` class

For the geolocation-policy document to be useful, the users will need to fine-tune it to their privacy requirements. This functionality is achieved by invoking the methods of the `GeolocationPolicy` instance. For instance, the `addGeodeticLocationToRule` method of the `FriendFinder` class can be used to add a new geodetic condition to any of the two default rules by invoking the `addGeodeticLocationToRule` method of its `GeolocationPolicy` instance. As it can be seen in Listing 6.4, the method takes in arguments identifying the rule (`closeties` or `looseties`) and a unique name (`placeName`) describing the location being added. We explained in the previous chapter that we have limited ourselves to expressing geodetic location conditions using Geoshape circles only. Therefore, we provide, as a parameter, an instance of the `Circle` class to the method. The method invokes the `addGeodeticLocationToRule` method of the `GeolocationPolicy` object in line 3 and indicates the success of the operation in line 6.

Other methods of the `GeolocationPolicy` object can be invoked by the developer in the same way as the `addGeodeticLocationToRule` method. The developer can access the

responses of the XCAP operations by using the `getResponse` method or the `GeolocationPolicy` object as done in line 4 of Listing 6.4. Developers could also implement the `xcapResponseHandler` method of the `NotificationListener` interface for the same purpose. In the next section, we discuss the implementation of the last two use cases.

6.2.3 Adding, Viewing and Removing Friends

We have so far discussed the implementation of the functionality of the friend-finder use cases apart from the *View, Add or Remove Friend* and *Send Friend Request* use cases. The *View, Add or Remove Friend* use case could have been handled by using IETF resource-lists. However, resource-lists are out of the scope of our work. Instead, we added the IETF Watcher-Info Event-Template Package (*watcher-info*) specified in [60] to the JME LBSs SDK prototype to implement the requirements stated in both use cases.

6.2.3.1 Adding Watcher Information Support to the Location-Based Services Development Kit Prototype

The watcher-info package maintains the subscription details pertaining to event publishers such as Targets [60]. Every publisher may subscribe to their watcher-info to determine who is interested in the information they publish. The MPS, by default, supports the watcher-info package. For each publisher, it stores the event type of the publication; the URI of the subscriber and their subscription status.

Listing 6.5 shows a sample watcher-info document. The document contains the subscription information of Target *sip:zelalem@ru.ac.za*. The watchers with the `status` attribute of the `watcher` element set to `pending` may be treated as having made a friend request to the Target. The user at the Target can add them to one of the `rules` in the geolocation-policy document to accept the friendship. The watchers with the `status` attribute set to `active` have already been authorised by the Target, *sip:zelalem@ru.ac.za*, to see the location information.

```
1 <?xml version="1.0"?>
2 <watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo" version="0"
    state="full">
3  <watcher-list resource="sip:zelalem@ru.ac.za" package="location">
4    <watcher id="77ajsyy76" event="subscribe" status="pending">
```

```
5       sip:shange@ru.ac.za
6     </watcher>
7     <watcher id="77ajsyy76" event="subscribe" status="active">
8       sip:mosioua@ru.ac.za
9     </watcher>
10  </watcher-list>
11 </watcherinfo>
```

Listing 6.5: A simple watcher-info document

Our JME LBSs SDK prototype already provides support for the SIP subscription process using the `Subscriber` class discussed in Chapter 4. Adding watcher-info support requires the reuse of an instance of this class to subscribe to the watcher-info of the `Target` instance of the application. However, it is important to distinguish between location and presence subscriptions from watcher-info subscriptions. The `beginSubscriptionTask` method of the `SipConnectionManager` class is modified to take in an event type parameter for this purpose. The event type of `presence` is used for location and presence subscriptions, while that of `presence.winfo` is used for the watcher-info subscriptions. Notifications on watcher-info subscription dialogs result in the content being parsed using methods of the `WatcherInfoParser` class that we created and discussed in Chapter 4. The methods of the class return a collection of `Watcher` instances to the `SipConnectionManager` class.

6.2.3.2 The Friend Request and Acceptance Sequence

The process of a Target subscribing to its watcher-info yields important information about friend requests. We need an architecture that uses this information to add the LC to the geolocation-policy documents of the Target.

Figure 6.5 illustrates the process of `FriendFinder` *A* making a friend request to `FriendFinder` *B*. *A* first adds *B*'s address to one of his default geolocation-policy rules at the XDMS. In step 3, *A* subscribes to *B*'s location information from the modified MPS. Because *A* is not in *B*'s geolocation-policy rules, the subscription status is set to `pending` at the server reflecting in *B's* watcher-info. When *B* subscribes to her watcher-info document she discovers that *A* has a `pending` subscription status and treats this as a friend request (steps 3 and 4). If *B* accepts the friend request, she adds *A* to one of her geolocation-policy rules as well (steps 7 and 8). Contrarily, if *B* opts to not grant the privilege of friendship

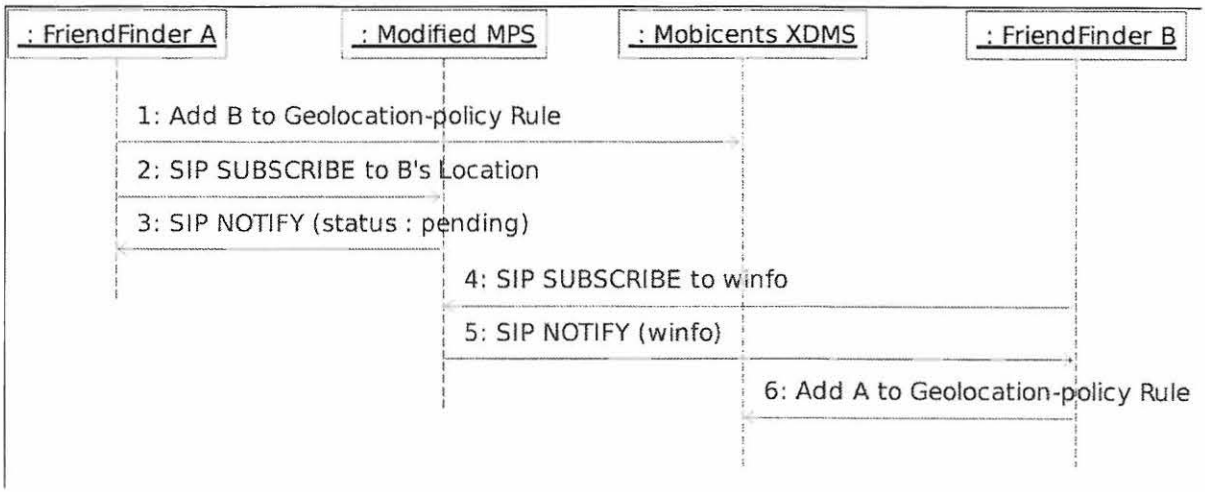

Figure 6.5: The Friend Request and acceptance sequence

to the request, then A's location information will be visible at B until the end-user at A decides to reverse the request by removing B from his buddy list and geolocation-policy documents. We did not implement an explicit mechanism to let A know that B has not accepted their friendship request. This can be noticed by the end-user at A because no location information will be available for B.

```
1 public void addFriend(String watcher, String rule)
2 {
3   super.addWatcherToRule(watcher, rule);
4   sipConnectionManager.addSubscriptionTask(watcher);
5 }
```

Listing 6.6: A portion of the `processFriendRequest` method of the `FriendFinder` class

6.2.3.3 Implementing Friend Request and Acceptance in the Location-Based Services Development Kit Prototype

The functionality required for the friend request and acceptance process, is added to the toolkit prototype by creating the `addWatcherToRule` method to the `GeolocPolicyAgent` abstract class. These method invokes the `addUserToRule` method of its `Geolocation-Policy` instance of the `FriendFinder` class. The rule name and the SIP address of the friend to be added to the rule are provided as parameters to both these methods. Classes

that extend the `GeolocPolicyAgent` class such as the `FriendFinder` class must call the `addUserToRule` method of this superclass. The `addSubscriptionTask` method of the `SipConnectionManager` instance of the `FriendFinder` class must then be called to start a subscription dialog with a `pending` status to the Target.

The process of accepting friend requests involves a Target subscribing to its watcher-info before invoking the `addUserToRule` and the `addSubscriptionTask` methods. Since this is a function that is required by all Targets, we add the `addWinfoSubscriptionTask` method to the `Target` interface to serve as a guide application implementation. We also added the `getWatcherInfo` method to guide Targets in polling for content on watcher-info subscriptions. Since `Target` instances such as the `FriendFinder` class can implement the `NotificationListener` interface, notifications on watcher-info subscriptions dialogs can be delivered to theses applications as soon as they are available at the mobile device.

```
1  public Vector getWatcherInfo()
2  {
3    return sipConnectionManager.getWatcherInfo();
4  }
5  public String getFriendInformationAsString()
6  {
7   String friendInformation = "";
8   Vector watcher = getWatcherInfo();
9   Enumeration tasks = watcher.elements();
10  while (tasks.hasMoreElements())
11  {
12    Watcher wat = (Watcher) tasks.nextElement();
13    friendInformation += "Address: " + wat.getSipAddress() + " Status
        : " + wat.getStatus() + "\n";
14  }
15  return  friendInformation;
16 }
```

Listing 6.7: The `getWatcherInfo` and `getFriendInformationAsString` methods of the `FriendFinder` class

Listing 6.7 shows the implementation of the `getWatcherInfo` method by the `FriendFinder` class. It also shows how the `getFriendInformationAsString` method invokes the `getWatcherInfo` method and puts the friend information in a string.

6.2.3.4 The Friend-Finder Midlet

The last piece of the friend-finder demonstration involves putting together a `FriendFinderMidlet` class. In JME applications, `Midlets` are used for application life-cycle management. They are also commonly used for managing application states and implementing mobile application user interfaces. The `FriendFinderMidlet` creates a `FriendFinder` instance upon loading and invokes its methods to perform friend-finder service use cases. We concluded the implementation of the service with this simple `Midlet`. In the next section, we look at the development of the child-tracker service.

6.3 The Child-Tracker Service

The child-tracker service, as we introduced it at the beginning of this chapter, is a service that is meant to enable parents to be notified when their children leave designated areas, such as playgrounds or parks. Messages may be pushed to the parent and to other concerned parties upon the occurrence of such an event. The child-tracker service requires location event notification filtering which, as discussed in the literature survey in Chapter 2, is a useful facility in realising push LBSs. The initial step towards the creation of this demonstration is to look at the general requirements of the service.

6.3.1 Application Requirements

Unlike the friend-finder service, the child-tracker service does not require peer applications mutually sharing location and presence information. Instead, the mobile devices belonging to the parents and other trusted parties host an LC applications known as `ParentMinders`. Another mobile device for the child hosts a Target application known as a `Child`. The `Child` application, shown in Figure 6.6, is responsible for three use cases. These are the *Register To Mobicents Registrar*, *Publish Location* and *Create or Modify Privacy Policy* use cases. These use cases were part of the friend-finder application as well. The *Create or Modify Privacy Policy* use case involves the `Child` creating or modifying privacy policies regarding access to their location information by the `ParentMinder` application.

The `ParentMinder` application is responsible for fulfilling two use cases. The *Register To Mobicents Registrar* use case is done in a similar fashion as to the friend-finder application.

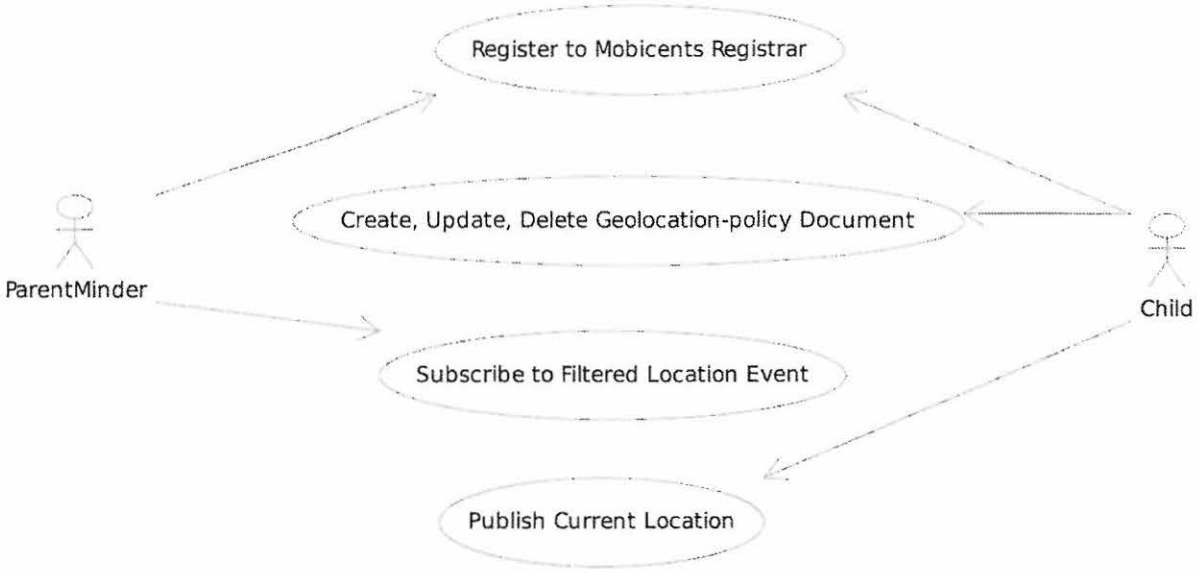

Figure 6.6: The child-tracker service use cases

The *Subscribe to Filtered Location Event* use case involves a `ParentMinder` subscribing to a specific location event that could be generated by a `Child` (leaving the designated area).

6.3.2 Implementing the Service by using the Location-Based Services Development Kit Prototype

Figure 6.7 shows the diagram of the main classes used in the child-tracker service. It also shows the relationships between these classes. We provided a discussion on the implementation of the *Register To Proxy. Publish Location* and the *Create or Modify Privacy Policy* use cases for the friend-finder application. For this reason, we do not discuss their implementation for the child-tracker service.

The most important implementation aspect to note here is that the `Child` application extends the `GeolocationPolicyAgent` abstract class because it works with geolocation policies (see 6.7). The *Subscribe to Filtered Location Event* use case requires the addition of location event notification filtering to both the JME SDK and to the modified MPS. For this reason, we discuss the addition of location notification event filtering to our environments first and the addition of this functionality to the child-tracker service later.

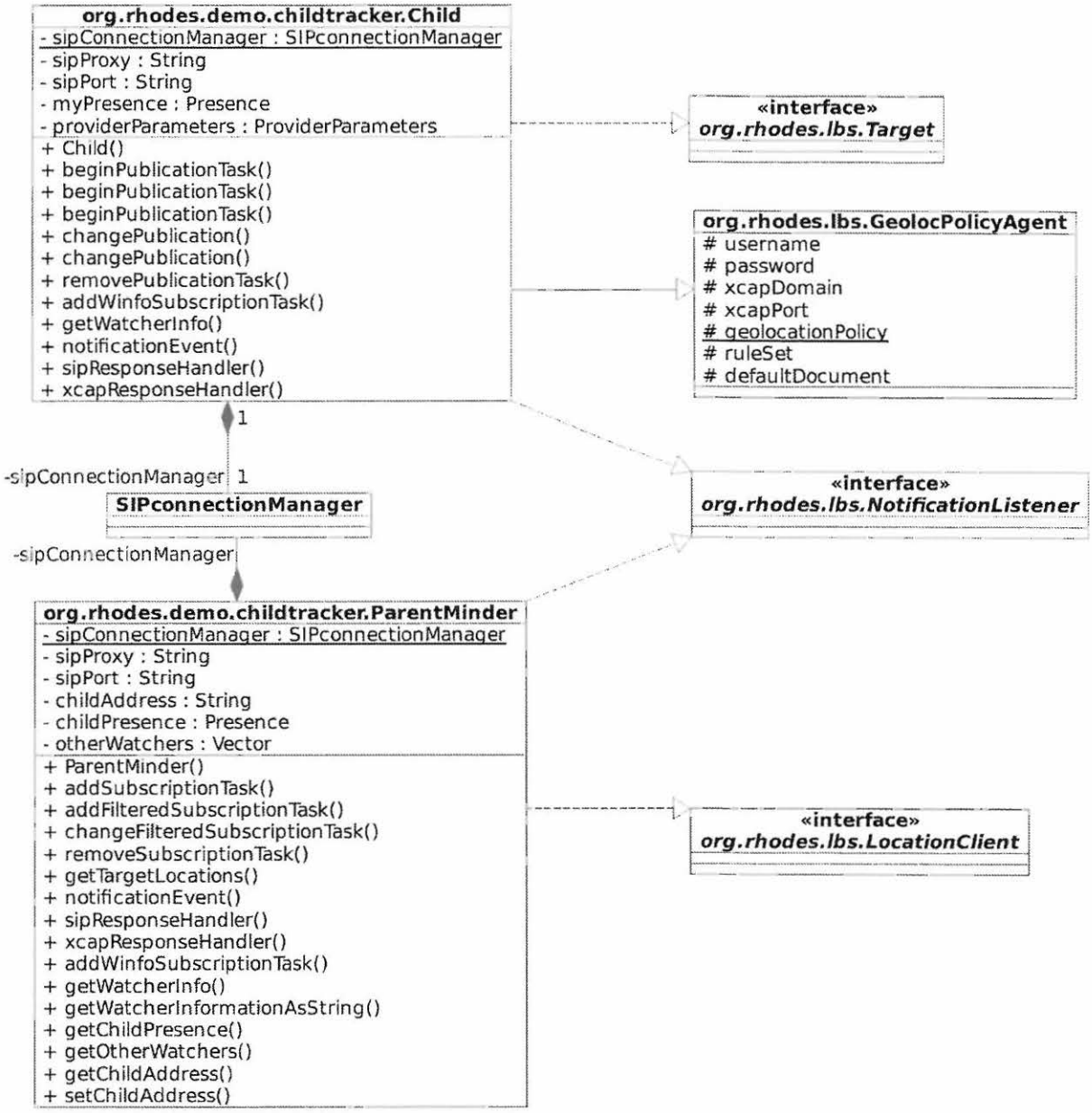

Figure 6.7: The classes used in the child-tracker service

6.3.3 Adding Location Event Notification Filtering to the Environments

We discussed location event notification filtering in Chapter 2 as a process that allows LCs to specify which location events they are interested in. This directly results into an LC getting only notifications that are relevant to them from a particular Target. We also mentioned that, according to [78], there are several location event filters that an LC may specify to trigger notifications from a Target.

While all the different location event filters may be useful in the implementation of different LBSs, we implemented only the *enterOrExit* filter as part of our project. This filter involves notifying an LC when a Target has entered or exited a given Geoshape circle. This type of filter was chosen because of its usefulness in realising the child-tracker service. In future, other filters may be added to the toolkit and MPS by following this demonstration.

6.3.3.1 Implementation of EnterOrExit Filtering on Location-Based Services Development Kit Prototype

Location event notification filtering functions by comparing the currently published PIDF-LO document (containing the current location and presence) of the Target against the criterion specified in the filter document provided by the LC. The *enterOrExit* filtering mechanism we are using checks if the currently published point is within the Geoshape circle given in the notification filter provided by the LC. A sample document containing the filter criterion of an LC is shown in Listing 6.8. The Geoshape circle is provided within the *enterOrExit* element.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <filter-set xmlns="urn:ietf:params:xml:ns:simple-filter"
3  xmlns:lf="urn:ietf:params:xml:ns:location-filter"
4  xmlns:gml="http://www.opengis.net/gml"
5  xmlns:gs="http://www.opengis.net/pidflo/1.0">
6  <filter id="123" uri="sip:presentity@example.com">
7   <trigger>
8    <lf:enterOrExit>
9     <gs:Circle srsName="urn:ogc:def:crs:EPSG::4326">
10     <gml:pos>42.5463 -73.2512</gml:pos>
11     <gs:radius uom="urn:ogc:def:uom:EPSG::9001">
```

```
12          850.24
13        </gs:radius>
14      </gs:Circle>
15    </lf:enterOrExit>
16  </trigger>
17 </filter>
18 </filter-set>
```

Listing 6.8: An example of an *EnterOrExit* filter using a circle

The filtering document is communicated to the MPS by an LC using SIP SUBSCRIBE requests. We modified the subscription part of the JME toolkit prototype to accommodate this requirement. The `addFilteredSubscriptionTask` and `changeFilteredSubscriptionTask` methods were added to the `SipConnectionManager` class to invoke the methods that construct and send the filtering documents in the `Subscriber` class. The `Subscriber` class was adapted to optionally send SUBSCRIBE requests with content (filter document) as it is done in the `Publisher` class for PIDF-LO documents. A service developer must implement the `LocationClient` interface that contains the `addFilteredSubscriptionTask` and `changeFilteredSubscriptionTask` methods.

6.3.3.2 Implementation of EnterOrExit Filtering on the Mobicents Presence Service: Adding Notification Filtering Fields to the Database

The MPS was originally built for presence and it does not support any form of notification filtering. We had to modify the location-aware MPS such that it receives notification filters (documents) from the LCs and determines whether the current location information of the Targets should result in notifications to the interested LCs. The currently published location of the Target should be located outside the Geoshape Circle given in the filter of the LC for notifications to be made.

As mentioned before, information pertaining to all current subscription dialogs is maintained in a database using the Hibernate JPA implementation on the MPS. We added the `DOCUMENT` and `CONTENT_TYPE` fields to the `Subscription` class which is an entity class for the JPA. This class creates the fields and persists the subscription data to the database during SIP transactions. The `SubscriptionControlSBB` that we discussed in the Chapter 3 receives the incoming SIP SUBSCRIBE requests. Through instances of helper classes, it maintains the state of `Subscriber` instances.

The two important helper classes for the `SubscriptionControlSBB` are the `NewSipSubscriptionHandler` and the `RefreshSipSubscriptionHandler`. The `NewSipSubscriptionHandler` helper class is responsible for creating new `Subscriber` instances. We added functionality to this class so that it stores the filtering document and its content type into the `DOCUMENT` and `CONTENT_TYPE` fields, respectively. The `RefreshSipSubscriptionHandler` helper class refreshes `Subscriber` instances after receiving subscription refresh requests. We refrained from adding functionality to this class so that it updates the `DOCUMENT` and `CONTENT_TYPE` fields upon such refresh requests. This limits the location-aware MPS to accept filters at the beginning of a subscription dialog only.

6.3.3.3 Implementation of EnterOrExit Filtering on the Mobicents Presence Service: Performing Location Event Notification Filtering

Having added the fields to store the filtering documents to the database, the final part of the building the service involved ensuring that these documents affect the location notification process. We discussed in Chapters 4 and 5 the use of JAXB bindings in the creation and parsing of XML documents at the MPS. Here, we create JAXB binding classes for notification filtering documents so that the `PresenceSubscriptionControlSBB` instances can unmarshal (parse) the filtering content of subscription requests. The filtering content is stored in the the `DOCUMENT` field of `Subscriber` instances created in 6.3.3.2. We put the auto-generated JAXB classes into the same package as the other JAXB classes generated to parse and compose PIDF-LO documents.

The second step involved making changes to the `SipSubscriberNotificationHandler` class (a helper class of the `SubscriptionControlSBB`), which handles the SIP notifications for the MPS. We modified the `createAndSendNotify` and the `notifySipSubscriber` methods so that before notifications to subscriptions that have filtering documents are made, the `filterNotificationsPerSubscriber` method is invoked. This is a method we created and added to the `PresenceSubscriptionControlSBB` to check if the *EnterOrExit* conditions are fulfilled. The method takes in the subscription dialog details, including the unmarshalled subscription content (the JAXB objects containing the contents of the filter) as parameters. It also takes in the currently published information (also unmarshalled) of the Target obtained from the `PresencePublicationControlSBB`. In the `filterNotificationsPerSubscriber` method of the `PresenceSubscriptionControlSBB`, the `getIsInArea` method of the `CircleConditon` class we created in Chapter 5

is used. As we discussed in Chapter 5, the `getIsInArea` method checks if the currently published point is within a given Geoshape circle. Notifications are made to the LC if the published point is outside the given Geoshape circle. As we mentioned earlier, the EnterOrExit mechanism we implemented only works when the given area is a Geoshape circle. Other geodetic shapes can be added in future.

This concludes the discussion on location notification filtering. In the next section, we discuss how this new functionality is used in the remaining case of the child-tracker service.

6.3.4 Implementing the Location Notification Filtering as part of the Child-Tracker Service

The `ParentMinder` class of the child-tracker service implements the `LocationClient` interface which guides the implementation of LC entities. The `ParentMinder` class implements the `addFilteredSubscriptionTask` method to be able to invoke its `SipConnectionManager` instance to perform filtered subscriptions. The signature of this method can be seen in Figure 6.7. The method takes in the SIP address of the `Child` and the Geoshape circle from which the `Child` should stray for notifications to be sent to the LC (filter criterion). The class also implements the `removeSubscriptionTask` for removing subscription dialogs to the child being watched. When the child leaves the given Geoshape circle, location notifications sent from the MPS can be received using the `notificationEvent` method of the `ParentMinder` class. The `notificationEvent` method is implemented as part of the `NotificationListener` interface.

This concludes the discussion of the implementation of the child-tracker service. Readers and developers can gain a deeper understanding of the demonstration by referring to the Javadoc provided or to the source code given in the `org.rhodes.demo` package.

6.4 Chapter Summary

This chapter described how we put together an abstract class and interfaces that can be used to aid developers in creating LBSs using the components created in Chapters 4 and 5. These classes and interfaces are provided in the `org.rhodes.LBSs` package and are meant to improve the developer-friendliness of the JME LBSs SDK prototype. We have also provided a discussion of two simple services that could be put together using the JME

LBSs SDK prototype and the modified MPS. A friend-finder service is a social networking service that allows friends to locate each other. A child-tracker service is one that allows parents to be notified only when their children leave designated areas, such as playgrounds or parks. Apart from demonstrating the use of the JME LBSs SDK prototype and MPS, new functionality was added during the implementation of these two services. The friend-finder services required the addition, removal and viewing of LCs that are interested in the location information of a particular Target. Such functionality was added to the JME LBSs SDK prototype. The child-tracker service had a requirement for location notification filtering. This function is important in realising many push LBSs as discussed in Chapter 2.

This concludes the development of the JME LBSs SDK prototype and the modified MPS and no more functionality was added as part of this project. In Appendix A, we provide a description of our JME LBSs SDK and the modified MPS from a service developer's point of view and highlight important issues regarding their usage.

Chapter 7

Conclusion

The increase in the number of smart mobile devices and the rapid growth of broadband services on mobile operator networks has resulted in the entrance of new Internet-based competitors in the telecommunications market. MNOs are under growing pressure to deliver new innovative multimedia services more quickly and efficiently. Although MNOs have several SDPs and SCEs at their disposal for the quick development of multimedia services, the location dimension has largely been ignored. Eventhough LBSss are not a significant generator of revenue for many MNOs, recent trends still indicate the growth of both the user-base and revenue of the LBSs market.

The present project was designed to put together a client-side JME LBSs SDK and a Mobicents LBSs server to aid MNOs and developers alike in the quick creation of the transport and privacy facilities of cross-referencing LBSs applications. The goals of our research as stated at the beginning of the thesis were as listed below:

1. Put together a Mobicents location service (composed of SBBs) that can be used for the sharing of cross-referencing LBSs information amongst different client entities. This would allow Target entities to publish their location information and for LCs to subscribe to it. The SBBs that make up the location server should be reusable and extensible to support the quick creation location-aware multimedia services.

2. Build a JME SDK for the quick development of client-side cross-referencing LBSs applications. The SDK will be used to develop Target and LC applications that share location information using the Mobicents location server.

3. LCs may be applications running on the mobile phones of close friends, family members or servers of marketing and government organisations. Therefore, Targets should be able to control which LCs see their location information and how it is shared amongst those LCs. The third goal involves was to put together facilities to ensure the privacy of the sensitive information shared by the Targets.

Even though we have met the goals stated above, there are limitations in the final products that are discussed later in section 7.3. This chapter concludes our work by discussing our final products and highlighting the contributions and the limitations of the project. At the end, we recommend further research that could be conducted in future.

7.1 A Summary of the Development Process and the Location-Based Services Tools

This project is part of a bigger research effort to put together a fully-functional convergence testbed based on the Mobicents SCE at Rhodes University. This mandated the use of the Mobicents SCE. As discussed in section 2.2, we considered several application layer protocols such as XMPP for the transport of location information amongst network entities but resolved to using SIP in the end. The decision was based on the fact that the document format for the transport of location information on SIP networks is not only rich but also requires less effort to be integrated into the Mobicents SCE which already had a SIP MPS. For the privacy protection of the location information, a decision to use geolocation-policies was made, as described in section 2.3. Geolocation-policies give more emphasis to the way information is shared amongst different network entities. In our opinion, this makes these policies more suitable to cross-referencing LBSs than other privacy protection approaches such as pseudononymisation and anonymisation.

The features of the JME LBSs SDK were incrementally put together to help developers with Target and LC applications capable of sharing location and basic presence information across SIP networks. The transport facilities of the SDK are exposed as part of one class that could be instantiated by LBSs developers to perform cross-referencing LBSs operations. Developers also have at their disposal interfaces that they could implement to guide them in the implementation of Target and LC applications using the SDK.

Our work on Mobicents led us to modify the MPS so that it may be responsible for the

management of the location and basic presence information published by Targets. The modified MPS distributes the location information to subscribing LCs by consulting and applying a geolocation-policy that is specified by the Target entities.

Apart from the LBSs transport features, the JME LBSs SDK was designed to aid the development of applications that may be used to create, modify and delete geolocation-policy documents located at the XDMS. Geolocation-policies provide privacy protection to the location information published by the Targets. The privacy functionality of the SDK is also exposed to developers as part of one class. Every application that uses these privacy facilities must use the provided abstract class to perform the universal geolocation-policy operations. This concludes the summary and, next, we highlight the main thesis contributions.

7.2 Thesis Contributions

7.2.1 Client and Server Tools for Creating Cross-Referencing Location-Based Services

There are several SDKs and APIs available on the Internet for the quick creation of self-referencing LBSs. Examples that fall in this category are the Garmin LBSs toolkit; the Nutiteq Mapping SDK; and the Navteq APIs and SDKs such as DeCarta. These SDKs provide functionality for querying informative content related to the particular location of a mobile device such as geocoding or navigation. For MNOs quickly to create more innovative multimedia applications with an LBSs dimension, SDKs are required that are not only self- but also cross-referencing. On the one hand, self-referencing toolkits will allow for content and mapping services to be included in LBSs applications. On the other hand, cross-referencing toolkits such as the JME LBSs SDK and the modified MPS will allow users to share their location information over the Internet according to their privacy preferences.

Furthermore, the focus of most cross-referencing implementations surveyed in our literature such as the push-based LAM and the LaPoC Services implemented in the IMS (section 2.2.6.4), is server-side. There seems to be a shortage of development toolkits for the quick creation of client-side SIP-based entities for cross-referencing LBSs such as Targets and LCs. Even though there are SCEs for server-based applications, it requires effort and

time to understand the specifications according to which client applications must be built. Developers could still face challenges in developing client-side applications that work well with server applications. Our JME LBSs SDK supports the quick creation of these client-side LBSs entities that may be compatible with SIP servers that are built according to SIP location transport specifications.

7.2.2 Client and Server Tools for Protecting Cross-Referencing Location-Based Services Privacy

Even though our focus from the beginning was on the Mobicents SCE and JME, we could not find any complete SDKs or implementations that supported the private sharing of location information according to geolocation-policies. Most of the literature surveyed discusses approaches such as anonymisation and pseudononimisation that are mostly suitable for self-referencing LBSs. Other applications such as Google latitude are based on XMPP and have a primitive on or off approach to privacy, where a user has an option to either share their location or not. This approach may limit the use of these services because the LCs are not always equally trusted by the Target. With geolocation-policies the Target publishes their location once to LCs that could range from very close friends to marketing organisations. The Target just needs to specify what their privacy preferences regarding each LC are. The MPS distributes the published location information according to the provided privacy policy.

7.2.3 Abstraction of Complex Protocols in Location-Based Services Development

The tools put together as part of this project abstract the complexities that come with SIP event signaling and the XCAP protocol. LBSs developers do not require the knowledge of these protocols to develop working applications. Instead of spending time understanding SIP as well as XCAP specifications, developers can set up the modified MPS and focus on the core aspects of their applications by using the JME LBSs SDK. Developers could also focus on using self-referencing SDKs and APIs available on the Internet to include mapping and other content in their LBSs applications.

7.3 Thesis Limitations

7.3.1 Lack of Independent Mobicents Location Service Building Blocks

The work done on the Mobicents involved the modification of the SIP MPS which was originally responsible only for the management and transmission of presence information. The modified MPS manages and transmits location and basic presence information for Targets and LCs. This is only a partial fulfillment of our first objective mentioned in section 1.4. Initially, the goal was to put together an independent Mobicents location server with SBBs that could be reused and extended to support the quick creation of location-aware multimedia services. We have not experimented with combining the modified MPS SBBs with those of other services such as media to provide rich server-side applications. This limitation dictates that location and presence applications are always combined because they are handled in the same SBBs. The lack of experimentation with the modified MPS results in a situation where most of the LBSs creation by developers is performed using the client JME LBSs SDK. We mention later that future work could focus on developing independent location SBBs that could be combined with other SBBs for the creation of rich multi-media services.

7.3.2 Incomplete Support of Location Formats

Section 4.5.1.1 discusses the implementation of the classes representing the location formats in the SDK. As mentioned in Chapter 4 section 4.5.1.1 only five of the seventeen civic location levels specified at [77] are supported by the JME LBSs SDK and the modified MPS. We also limited the geographical shapes that could be used only to two dimensional points, circles and polygons. Section 5.4.2 discusses the process of implementing private SIP subscriptions. This process can only be implemented if the geographical shape published by a Target is a geodetic point. Other geographical shapes can be published by the Targets without any privacy protection. Future work in this area could be focused on addressing these limitations.

7.3.3 Fixing the Java Micro Edition Session Initiation Protocol Bug and the Unsupported HTTP Methods

In section 4.2.2.1, we highlighted a problem resulting from the inclusion of SIP ports in all SIP requests made using the JME SIP library. The problem was fixed by modifying the Mobicents RA to strip the incorrectly included port. Also, section 5.3.1.1 emphasises the fact that JME does not support HTTP PUT and DELETE requests. These requests are central to XCAP protocol functionality. We implemented the XCAP client of the JME LBSs SDK to make it use modified HTTP POST requests to send PUT and DELETE requests. The Mobicents XDMS was modified to cater for this deviation from the standards. Both these work-arounds limit the compatibly of the JME LBSs SDK to other SIP servers that are built according to location standards but do not cater for this modification.

7.4 Recommendations for Future Research

The suggestions for future research stemming from this project are derived directly from the fore-mentioned limitations. The following are our recommendations for future research:

- The modified MPS and the JME LBSs SDK could be changed to become a location server that is independent of presence information processing. By doing this, the location server SBBs could be reused and extended to support the quick creation of location-aware multimedia services. The JME LBSs SDK could also be used quickly to add a location dimension to other mobile device applications.

- The JME LBSs SDK could be improved by integrating it with self-referencing LBSs toolkits and APIs such as the Nutiteq Mapping SDK for a more complete LBSs development solution. This integration will bring to the SDK services such as geocoding, reverse-geocoding, route planning, mapping and navigation. This will increase both the scope and usefulness of the SDK because the LBSs tools will be readily available to developers in a single toolkit.

- Other efforts could focus on making the JME LBSs SDK more complete. Unsupported geolocation shapes and civic location fields could be added to both the JME LBSs SDK and the modified MPS. The JME LBSs SDK could also be extended by implementing other SIP standards such as resource-lists.

- The location transport applications developed with the JME LBSs SDK should be compatible with any other SIP server that processes location information according to RFC 4119 of the IETF. This is assumed because the SDK and the modified MPS were put together according to IETF standards. The location privacy applications developed with the JME LBSs SDK should be compatible with any other XDMS server that supports geolocation-policies as specified in [66]. Future work could also focus on the actual testing and usage of the LBSs tools developed as a part of this project with other SIP servers and XDMSs to verify their comaptibility. The reader must be aware of the limitations such as the inclusion of ports in JME SIP requests before attempting to use the SDK with other SIP servers. These limitations are discussed in section 7.3.

Future efforts could be targeted towards improving and experimenting with the above mentioned aspects of the modified MPS as well as the JME LBSs SDK.

Bibliography

[1] Adams, P. The Real Life Social Network. Online: `http://thenoisychannel.com/2010/07/08/paul-adamss-presentation-on-social-networking/` [Last accessed: 28 February 2012], April 2010.

[2] Brajdic, A., Lapcevic, O., Matijasevic, M., and Mosmondor, M. Service Composition in IMS: A Location-Based Service Example. *International Symposium on Wireless Pervasive Computing* (2008), 208–212.

[3] Carmarillo, G., and Garcia-Martin, M. A. *The 3G IP Multimedia Subsytem (IMS), 3rd Edition.* Wiley, 2008.

[4] Chi-Yin, C., and Mohamed, M. F. Privacy in Location-Based Services: A System Architecture Perspective. *SIGSPATIAL Special 1* (July 2009), 23–27.

[5] Chua, G. Mobile Network at the Tipping Point: The Data Explosion and Next Generation Network Challenge. IDC and Juniper Networks, February 2010.

[6] Costa-Requena, J., and Tang, H. Application of Spatial Location Information to SIP. *Cluster Computing 5* (October 2002), 399–410.

[7] Dinsing, T., Eriksson, G. A., Fikouras, I., Gronowski, K., Levenshteyn, R., Pettersson, P., and Wiss, P. Service Composition in IMS Using Java EE SIP Servlet Containers. Online: `http://www1.ericsson.com/ericsson/corpinfo/publications/review/2007_03/files/4_ServiceComposition.pdf` [Last accessed: 12 May 2012], 2007.

[8] Dĺurr, F., Wernke, M., Skvorzov, P., and Rothermel, K. Towards a Position Sharing Approach for Location-based Services. *W3C Workshop on Privacy for Advanced Web APIs Paper 15* (2010), 1–2.

[9] ENOUGH SOFTWARE. *Mobile Developer Guide to the Galaxy, 6th Edition.* Enough Software, 2010.

[10] ERICSSON LABS. Developing an IMS SIP Client on Android. Online: `https://labs.ericsson.com/developer-community/blog/develop-ims-sip-client-android-part-2` [Last accessed: 12 May 2012], 2010.

[11] ERICSSON LABS. Locate End Users With Labs API. Online: `http://www.ericsson.com/news/100628_locate_users_244218601_c` [Last accessed: 12 May 2012], June 2010.

[12] FERRARO, R., AND AKTIHANOGLU, M. *Location Aware Applications.* Manning, 2010.

[13] FERRY, D. JAIN SLEE (JSLEE) 1.1 Specification, Final Release. Opencloud and Sun Microsystems, 2003-2008.

[14] FUJIMOTO, S. RFC3863: Presence Information Document Format (PIDF). Internet Engineering Task Force (IETF), August 2004.

[15] GABRIEL, G., PANOS, K., ALI, K., CYRUS, S., AND KIAN-LEE, T. Private Queries in Location-Based Services: Anonymizers Are Not Necessary. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2008), SIGMOD '08, ACM, pp. 121–132.

[16] GEOGRAPHIC LOCATION/PRIVACY (GEOPRIV). Location Services @ IETF72. Online: `http://sites.google.com/site/geopriv/` [Last accessed: 28 February 2012].

[17] GIBSON, B. Mobile Location-Based Services Market to Exceed USD 12bn By 2014 Driven By Increased Apps Store Usage, Smartphone Adoption and New Hybrid Positioning Technologies. Online: `http://juniperresearch.com/viewpressrelease.php?pr=180` [Last accessed: 12 May 2012], 2010.

[18] GSM ARENA. Android and iOS Growth Continues As RIM Falters, ComScore Reports. Online: `http://www.gsmarena.com/android_and_ios_growth_continues_as_rim_falters_comscore_reports-news-2857.php` [Last accessed: 12 May 2012], July 2011.

[19] HAMBLEN, M. Verizon Launches Toolkit for Location App Developers. Online: `http://www.computerworld.com/s/article/9187098/Verizon_launches_toolkit_for_location_app_developers` [Last accessed: 12 May 2012], September 2010.

[20] HARTMANN, C. J2ME REST Client. Online: `http://www.acidum.de/2008/12/29/j2me-rest-client/` [Last accessed: 12 May 2012], December 2008.

[21] HENG-TE, C., WEN-SHIUNG, C., YI-HUNG, H., AND JENG-YUENG, C. An Easy Way for Location-Based Services. In *IASTED European Conference on Proceedings of the IASTED European Conference: Internet and Multimedia Systems and Applications* (Anaheim, CA, USA, 2007), ACTA Press, pp. 23–27.

[22] HICKS, J., CHRISTIAN, A., AND AVERY, B. Integrating Presence and Location Services using SIP. *HP Labs Tech Report* (2005).

[23] ISOMAKI, M. RFC 4827: The Extensible Markup Language (XML) Configuration Access Protocol (XCAP). Internet Engineering Task Force (IETF), May 2007.

[24] JOE HILDEBRAND, P. S.-A. XEP-0080: User Location, 2009.

[25] KÜPPER, A., AND TREU, G. From Location to Position Management: User Tracking for Location-based Services. In *KiVS Kurzbeiträge und Workshop* (2005), P. Müller, R. Gotzhein, and J. B. Schmitt, Eds., vol. Vol. 61 of *LNI*, GI, pp. 81–88.

[26] KÜPPER, A., TREU, G., AND LINNHOFF-POPIEN, C. TraX: A Device-Centric Middleware Framework for Location-Based Services. *Communications Magazine, IEEE Vol. 44* (2006), 114–120.

[27] LEVENT-LEVI, T. SIMPLE vs. XMPP Showdown. Online: `http://blog.radvision.com/voipsurvivor/2008/07/17/simple-vs-xmpp-showdown/` [Last accessed: 12 May 2012], July 2007.

[28] LIM, S., O'DOHERTY, P., FERRY, D., AND PAGE, D. JAIN SLEE Tutorial - Introducing JAIN SLEE. Online: `http://www.jainslee.org/downloads/jainslee-tutorial-04.pdf` [Last accessed: 12 May 2012], 2003.

[29] LOUISE BARKHUUS. Privacy in Location-Based Services, Concern vs. Coolness. *Mobile HCI 2004 Workshop: Location System Privacy and Control* (2004).

[30] LOYTANA, K. JSR 179: Location API for J2ME. Java Specification Requests, March 2006.

[31] MAHMOUD, Q. H. J2ME Low-Level Network Programming with MIDP 2.0. Online: `http://developers.sun.com/mobility/midp/articles/midp2network/` [Last accessed: 12 May 2012], April 2003.

[32] MAHMOUD, Q. H. Getting Started with SIP API for J2ME (JSR 180). Online: `http://developers.sun.com/mobility/apis/articles/sip/` [Last accessed: 12 May 2012], November 2004.

[33] MAHMOUD, Q. H. J2ME and Location-Based Services. Online: `http://developers.sun.com/mobility/apis/articles/location/` [Last accessed: 12 May 2012], March 2004.

[34] MARTINS, E. Mobicents SIP Presence Service: XDM Server Creating XCAP Application Usages. Online: `http://www.google.com/url?sa=D&q=https://mobicents.googlecode.com/svn/trunk/servers/sip-presence/docs/misc/MobicentsSIPPresenceService_XDMS_CreatingXcapAppUsages.ppt` [Last accessed: 12 May 2012], August 2010.

[35] MARTINS, E., BARANOWSKI, B., AND MENDONÇA, A. Mobicents JAIN SLEE Sip Service Example User Guide. Red Hat, Inc., 2010.

[36] MARTINS, E., AND SHANGE, N. Adding A New AppUsage to the XDMS, August 2010.

[37] MIKE LOUSHINE, D. L. Location Conveyance with IMS: the OMA LOCSIP Service Enabler.

[38] MOSMONDOR, M., SKORIN-KAPOV, L., FILJAR, R., AND MATIJASEVIC, M. Conveying and Handling Location Information in the IP Multimedia Subsystem. *Journal of Communications Software and Systems vol. 2 no. 4* (2006), 313–321.

[39] NDAKUNDA, S., TERZOLI, A., AND WRIGHT, M. Adding Location-Based Services Support to the Mobicents Service Delivery Platform. In *Southern African Telecommunication Conference (SATNAC)* (2010), pp. 210–215.

[40] NDAKUNDA, S., TERZOLI, A., AND WRIGHT, M. Composing a Simple Friend-Finder Application Using the SIP Location-Based Services Toolkit. In *Southern African Telecommunications Conference (SATNAC)* (2011), pp. 210–216.

[41] NILE TECHNOLOGIES. Telecom - Service Delivery Platform (SDP). Online: `http://www.niletechnologies.com/Telecom-SDP.html` [Last accessed: 12 May 2012], December 2010.

[42] NOKIA CORPORATION. Interface SipClientConnection. Online: `http://library.forum.nokia.com/index.jsp?topic=/Java_Developers_Library/GUID-2508C2ED-C0BE-4512-9302-6805AB7ACB0E/javax/microedition/sip/SipRefreshListener.html` [Last accessed: 12 May 2012], 2004.

[43] NOKIA CORPORATION. Interface SipRefreshListener. Online: `http://library.forum.nokia.com/index.jsp?topic=/Java_Developers_Library/GUID-2508C2ED-C0BE-4512-9302-6805AB7ACB0E/javax/microedition/sip/SipRefreshListener.html` [Last accessed: 12 May 2012], 2007.

[44] NORD, J., SYNNES, K., AND PARNES, P. An Architecture for Location Aware Applications. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)* (Washington, DC, USA, 2002), vol. 9 of *HICSS '02*, IEEE Computer Society, pp. 293–.

[45] OPEN MOBILE ALLIANCE. TS 24.141: Location in SIP/IP core - Candidate Version 1.0. Online: `http://cms.comsoc.org/SiteGen/Uploads/Public/Docs_Globecom_2009/4-LOCSIP-Overview-lukacs_loushine.pdf` [Last accessed: 10 April 2011], August 2009.

[46] ORT, E., AND MEHTA, B. Java Architecture for XML Binding (JAXB). Online: `http://www.oracle.com/technetwork/articles/javase/index-140168.html` [Last accessed: 12 May 2012], March 2003.

[47] PAAVOLAINEN, S. Fast implementation of RSA's MD5 hash generator in Java JDK Beta-2 or higher. Online: `http://blackberry.svn.wordpress.org/trunk/src/com/wordpress/utils/MD5.java` [Last accessed: 12 May 2012], 1996.

[48] PAILER, R., WEGSCHEIDER, F., AND BESSLER, S. A Terminal-Based Location Service Enabler for The IP Multimedia Subsystem. In *WCNC* (2006), IEEE, pp. 2285–2290.

[49] PETERSON, J. RFC4119: A Presence-based GEOPRIV Location Object Format. Internet engineering Task Force (IETF), Decmber 2005.

[50] PETTEY, C., AND GOASDUFF, L. Gartner Identifies 10 Consumer Mobile Applications to Watch in 2012. Online: `http://www.gartner.com/it/page.jsp?id=1544815` [Last accessed: 12 May 2012], February 2011.

[51] PHILLIPS, D. What's New in Sun JavaŹ Wireless Toolkit 2.5 for CLDC. Online: `http://developers.sun.com/mobility/wtk/articles/sjwtoolkit2.5/` [Last accessed: 12 May 2012], February 2007.

[52] POLK, J. Location Conveyance for the Session Initiation Protocol draft-ietf-sip-location-conveyance-13. Internet Engineering Task Force (IETF), September 2009.

[53] POOLSAPPASIT, N., AND RAY, I. Towards Achieving Personalized Privacy for Location-Based Services. *Trans. Data Privacy 2* (April 2009), 77–99.

[54] REDHAT: MOBICENTS JAIN-SLEE. Mobicents Jain-Slee. Online: `http://www.mobicents.org/slee/intro.html` [Last accessed: 12 May 2012].

[55] REDHAT: MOBICENTS JAIN-SLEE. Mobicents SIP Presence Service Guide. Online: `http://hudson.jboss.org/hudson/job/MobicentsBooks/lastSuccessfulBuild/artifact/sip-presence/index.html` [Last accessed: 12 May 2012], 2008.

[56] REDHAT: MOBICENTS JAIN-SLEE. Mobicents. Online: `http://www.mobicents.org/products.html` [Last accessed: 12 May 2012], July 2010.

[57] ROACH, A. B. RFC3265: Session Initiation Protocol (SIP)-Specific Event Notification. Internet Engineering Task Force (IETF), June 2002.

[58] RODDEN, T., FRIDAY, A., MULLER, H., AND DIX, A. A Lightweight Approach to Managing Privacy in Location-Based Services. Tech. Rep. CSTR-07-006, University of Nottingham and Lancaster University and University of Bristol, October 2002.

[59] ROSENBERG, J. SIP: Session Initiation Protocol. Internet Engineering Task Force (IETF), June 2002.

[60] ROSENBERG, J. RFC 3857: The Watcher Information Template-Package for The Session Initiation Protocol (SIP) Event Framework. Internet Engineering Task Force (IETF), August 2004.

[61] ROSENBERG, J. Extensible Markup Language (XML) Formats for Representing Resource Lists. Internet Engineering Task Force (IETF), May 2007.

[62] ROSENBERG, J. RFC 5025: Presence Authorization Rules. Internet Engineering Task Force (IETF), December 2007.

[63] SANDFORD, B. A System for Locating Mobile Terminals With Tunable Privacy. *J. Theor. Appl. Electron. Commer. Res. 2* (August 2007), 82–91.

[64] SCHILLER, J., AND VOISARD, A. *Location Based Services*. Morgan Kaufmann, 1999.

[65] SCHULZRINNE, H. RFC 4480 RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF). Internet Engineering Task Force (IETF), July 2006.

[66] SCHULZRINNE, H. IETF Draft: Geolocation Policy: A Document Format for Expressing Privacy Preferences for Location Information. Geographic Location/Privacy (geopriv), Internet Engineering Task Force, October 2010.

[67] SCHULZRINNE, H., AND TSCHOFENIG, H. RFC 4745: Common Policy: A Document Format for Expressing Privacy Preferences. Internet Engineering Task Force, February 2007.

[68] SCHULZRINNE, H., TSCHOFENIG, H., NEWTON, A., AND HARDIE, T. LoST: A Protocol for Mapping Geographic Locations to Public Safety Answering Points. In *IPCCC* (2007), IEEE Computer Society, pp. 606–611.

[69] SHACHAM, R., SCHULZRINNE, H., KELLERER, W., AND THAKOLSRI, S. An Architecture for Location-Based Service Mobility Using the SIP Event Model. In *The International Conference on Mobile Systems, Applications, and Services* (2004), p. 3.

[70] SHEK, S., AND GRANTS, C. Next-Generation Location-based Services for Mobile Devices. Online: `http://assets1.csc.com/lef/downloads/CSC_Grant_2010_Next_Generation_Location_Based_Services_for_Mobile_Devices.pdf` [Last accessed: 12 May 2012], February 2010.

[71] SILAS, D., MARTINS, E., MORGAN, J., AND WELLS, T. SIP Presence Service User Guide. Red Hat, Inc., 2010.

[72] SINGH, V. K., SCHULZRINNE, H., BONI, P., ELMAN, B., AND KENNESON, D. Presence Aware Location-Based Service For Managing Mobile Communications. In *Consumer Communications and Networking Conference* (2007), pp. 514 - 519.

[73] SIP COMMUNICATOR. Geolocation Aware Contact List with Jabber. Online: `http://www.sip-communicator.org/index.php/Development/GeolocationAwareContactListWithJabber` [Last accessed: 12 May 2012].

[74] SOURCEFORGE. About kXML. Online: `http://kxml.sourceforge.net/about.shtml` [Last accessed: 12 May 2012], 2005.

[75] STEINIGER, S., NEUN, M., AND EDWARDES, A. Foundations of Location Based Services. Online: `http://www.spatial.cs.umn.edu/Courses/Fall11/8715/papers/IM7_steiniger.pdf` [Last accessed: 12 May 2012].

[76] SUN MICROSYSTEMS. Interface HttpConnection. Online: `http://java.sun.com/javame/reference/apis/jsr118/javax/microedition/io/HttpConnection.html` [Last accessed: 12 May 2012], 2006.

[77] THOMSON, M. RFC5139: Revised Civic Location Format for Presence Information Data Format Location Object (PIDF-LO). Internet Engineering Task Force (IETF), February 2008.

[78] TSCHOFENIG, H. IETF Draft: Filtering Location Notifications in the Session Initiation Protocol (SIP) draft-ietf-geopriv-loc-filters-11. Geographic Location/Privacy (GEOPRIV), Internet Engineering Task Force (IETF), March 2010.

[79] TSCHOFENIG, H., HENNING, S., NEWTON, A., AND PETERSON, J. The IETF Geopriv and Presence Architecture Focusing on Location Privacy. In *W3C Workshop on Languages for Privacy Policy Negotiation and Semantics-Driven Enforcement* (2006).

[80] VENESS, C. Calculate Distance, Bearing and More Between Latitude/Longitude Points. Online: `http://www.movable-type.co.uk/scripts/latlong.html` [Last accessed: 12 May 2012], January 2010.

[81] VERIZON. APIs and Toolkits. Online: `http://developer.verizon.com/content/vdc/en/verizon-platforms/verizon-mobile-internet-services/verizon-mobile-internet-apis-overview.html` [Last accessed: 12 May 2012], September 2010.

[82] VERVERIDIS, C., AND POLYZOS, G. C. Mobile Marketing Using Location Based Services.

[83] VINAYTECHS. XMPP and SIMPLE: A Comparative Study. Online: `http://vinaytechs.blogspot.com/2009/10/xmpp-and-simple-comparative-study.html` [Last accessed: 12 May 2012], October 2009.

[84] WHITE, J., AND HEMPHILL, D. *Java 2 Micro Edition.* Manning Publications Co., 2002.

[85] WILSON WU, ALEKSANDAR RADOVANOVIC, W. D. T. SIP-based Location Service Provision. In *Southern African Telecommunications and Networking Conference (SATNAC)* (2005), vol. 1, pp. 371–376.

[86] WU, X., AND SCHULZRINNE, H. Location-Based Services in Internet Telephony. In *2nd IEEE Consumer Communications and Networking Conference* (2005), IEEE, pp. 331–336.

[87] XU, H., AND TEO, H.-H. Consumers' Privacy Concerns toward Using Location-Based Services: An Exploratory Framework and Research Proposal. In *ECIS* (2005), D. Bartmann, F. Rajola, J. Kallinikos, D. E. Avison, R. Winter, P. Ein-Dor, J. Becker, F. Bodendorf, and C. Weinhardt, Eds., pp. 866–875.

Appendix A

An Overview of the Location-Based Services Tools and Guidelines for Using Them

In the chapters of the thesis, we discussed the building of the JME LBSs SDK and the modified MPS. Here, we discuss these tools from an LBSs developer's point of view. The reason is to provide guidelines on how the final tool set can be used to facilitate the development of LBSs. Developers do not need to know the SIP or XCAP protocols to use the tools. They need to be familiar only with the facilities provided and how to use them. Most of the development of the services is expected to happen on the client JME LBSs SDK which is designed to work with the modified MPS and XDMS infrastructure. For this reason, a large portion of this chapter discusses JME LBSs SDK facilities. In the final sections, we provide general guidelines and set-up instructions for the modified MPS and XDMS infrastructure available to LBSs developers.

A.1 Guidelines for Building Basic Location-Based Services and Setting Up the Location-Based Services Development Kit : Information Transport

This section provides service development information regarding the location and presence transport facilities of the JME LBSs SDK . Firstly, we discuss how the SDK is set up for

use. After this, we discuss what the SDK can and cannot do on behalf the the service developer and provide general guidelines that developers can follow to put together LBSs.

A.1.1 Setting Up The Location-Based Services Development Kit

The JME LBSs SDK can be used in projects developed using the Java ME 3.0 SDK or later. The Netbeans Integrated Development Environment (IDE) has distributions that have the SDK and mobile phone emulators by default. The SDK can also be installed separately. The developer should include the *sipLBSslib.jar* as one of the library files of their JME projects. This jar file contains the JME code and libraries needed to use all the facilities of the SDK. Developers can use the *toolkitdemos* project to see an example of how a *sipLBSslib.jar* file can be included. The developer can also see how the library functions can be imported and invoked by studying the demonstrations provided in this project. The *toolkitdemos* project contains the code for the two demonstration services discussed in Chapter 6. The toolkit is not limited to the Netbeans IDE, other Java-based IDEs that support JME may be used as well.

A.1.2 The `SipConnectionManager` Class and the Functionality it Provides

The `SipConnectionManager` class manages all the SIP connections in the JME LBSs SDK. As mentioned in the introduction, knowledge of the SIP protocol is not necessary in order to use the SDK. Services that require the transmission of location and presence information between Targets and LCs have to instantiate the `SipConnectionManager` class. To import the class the line `import org.rhodes.sipevent.SIPconnectionManager;` should be used. Examples of creating instances of the class can be found in the *toolkitdemos* project. There must be only one instance of the `SipConnectionManager` class per LBSs application. Figure A.1 shows the `SipConnectionManager` class and its public attributes and operations that can be used by developers. The functionality performed by each operation is described in the Javadoc provided with this thesis.

org.rhodes.sipevent.SIPconnectionManager
+ DISCONNECTED : int + REGISTERING : int + REGISTERED : int + PUBLISHING : int + PUBLISHED : int + PENDING : int + SUBSCRIBED : int + SUBSCRIBING : int + FAILED : int + reg : Registrar + pub : Publisher
+ SIPconnectionManager(userName : String, proxy : String, localPort : String, serverPort : String) + SIPconnectionManager(userName : String, proxy : String, localPort : String, serverPort : String, expires : String) + beginPublicationTask(status : String, note : String, location : org.rhodes.sipevent.pojo.Location) + beginPublicationTask(pres : org.rhodes.sipevent.pojo.Presence) + beginPublicationTask(interval : int, timeout : int, maxAge : int, note : String) + changePublication(status : String, note : String, location : org.rhodes.sipevent.pojo.Location) + changePublication(pres : org.rhodes.sipevent.pojo.Presence) + removePublicationTask() + addSubscriptionTask(to : String, event : String) + addSubscriptionTask(to : String) + addWinfoSubscriptionTask() + removeSubscriptionTask(to : String) + getSubscriptionContent(event : String) : Vector + getTargetLocations() : Vector + getWatcherInfo() : Vector + addNotificationListener(notificationListener : org.rhodes.lbs.NotificationListener) + notificationEvent(eventType : String, information : Vector) + notifyResponse(scc : SipClientConnection) + notifyRequest(scn : SipConnectionNotifier) + getCurrentContact() : String[] + locationUpdated(lp : LocationProvider, location : javax.microedition.location.Location) + providerStateChanged(lp : LocationProvider, i : int) + handleResponse(requestTyppe : String, statusCode : int, isError : boolean) + getContact() : String + getExpires() : String + getProxy() : String + getScn() : SipConnectionNotifier + getSipAddress() : String + getSubcriptionTasks() : Vector + getUserName() : String + setContact(contact : String) + setExpires(expires : String) + getLocalPort() : String + setLocalPort(localPort : String) + getServerPort() : String + setServerPort(serverPort : String) + setProxy(proxy : String) + setScn(scn : SipConnectionNotifier) + setSipAddress(sipAddress : String) + setSubcriptionTasks(subcriptionTasks : Vector) + setUserName(userName : String) + getLatestLocation() : javax.microedition.location.Location + setLatestLocation(aLatestLocation : javax.microedition.location.Location) + getLocations() : Vector + setLocations(aLocations : Vector) + getInterval() : int + setInterval(aInterval : int) + getTimeout() : int + setTimeout(aTimeout : int) + getMaxAge() : int + setMaxAge(aMaxAge : int) + getPub() : Publisher + getReg() : Registrar + setPub(aPub : Publisher) + setReg(aReg : Registrar) + getNotificationListener() : org.rhodes.lbs.NotificationListener + setNotificationListener(notificationListener : org.rhodes.lbs.NotificationListener)

Figure A.1: The `SipConnectionManager` class and its public attributes and methods

A.1.2.1 Guidelines for Implementing Target Entities

We mentioned in earlier chapters that a Target is an entity that generates and shares location information with other entities. Targets require the publication aspects of the `SipConnectionManager` class instances. The Target interface of the `org.rhodes.LBSs` package requires the implementation of the methods that can be used to initiate, change and stop SIP publication using `SipConnectionManager` instances. Developers should implement this interface to guide them in implementing Target entities.

`SipConnectionManager` class instances maintain SIP publication tasks for the developer. There can be at most one publication task being performed by a given mobile device. Both the `SipConnectionManager` class and the Target interface provide overloaded `beginPublicationTask` and `changePublication` methods for initiating and changing publications, respectively. The use of these methods is discussed in detail in Chapter 6. The signatures of these methods can be seen in Figure A.1. One of the overloaded `beginPublicationTask` methods of the `SipConnectionManager` class utilises the JME Location API and uses the most suitable location determination technology available to determine and publish location information. When this method is used, a developer does not need manually to provide location information. They need to provide only basic presence information. If other overloaded `beginPublicationTask` methods are used then the developer must parse location and presence information as parameters by creating appropriate instances of classes from the `org.rhodes.sipevent.pojo` package. The location information can either be civic or geodetic. The publication tasks must be removed by invoking the `removePublicationTask` method of the `SipConnectionManager` instance.

Target applications need information about the LCs that are interested in seeing their location and presence information. These enables Targets to authorise LCs and to control the privacy levels of the information shared with these LCs. The Target interface requires the implementation of the `addWinfoSubscriptionTask` method to start a watcher-info subscription dialog for the Target. These subscriptions are initiated by invoking the `addWinfoSubscriptionTask` method of the `SipConnectionManager` class.The interface also requires the implementation of the `getWatcherInfo` method to allow for Targets to poll for the watcher information obtained from the watcher-info subscription dialog. The `getWatcherInfo` method returns a collection of `Watcher` instances (from the class in the `org.rhodes.sipevent` package) containing the watcher information parsed from watcher-info documents. Instead of polling for watcher-info subscription information using the

`getWatcherInfo` method, developers can implement the `NotificationListener` interface to get this information as soon as it is available at the `SipConnectionManager`.

A.1.2.2 Guidelines for Implementing Location Client Entities

In Chapter 2 we mentioned that an LC is an entity that watches or consumes the location information of Targets. LCs must use the subscription aspects of the `SipConnectionManager` class instances. The `LocationClient` interface of the `org.rhodes.LBSs` package requires the implementation of methods that initiate and remove SIP subscription dialogs to given Targets using instances of the `SipConnectionManager` class. Developers should implement this interface to guide them in implementing applications for LC entities.

The `SipConnectionManager` class instances maintain SIP subscription dialogs for the developer. Each mobile device must have only one LC entity. This entity can initiate many subscription dialogs with specified Targets. Both the `SipConnectionManager` class and the `LocationClient` interface provide overloaded `beginSubscriptionTask` methods for initiating a subscription dialog with a single Target. The use of these methods is discussed in detail in the preceding chapter. The reader can also refer to the *toolkitdemos* project to see how the method could be used to start subscription dialogs. Subscription dialogs can be removed by invoking the `removeSubscriptionTask` method of the `SipConnectionManager` instance. Figure A.1 provides the signatures of the SIP subscription methods of the SDK.

The `LocationClient` interface requires the implementation of the `getTargetLocations` methods. This method can be used to poll for the location and presence information of the Targets that the LC subscribed to. As we discussed in the preceding chapter, the polling mechanism could be wasteful on limited mobile device resources. Instead, developers are advised to implement the `NotificationListener` interface for their LCs so that location and presence notifications are available to them immediately after the modified MIPS has sent them. The `notificationEvent` method of the `NotificationListener` interface receives only location and presence or watcher information. The developer has to differentiate between the two events as it can be seen in the *toolkitdemos* project.

A.1.2.3 The `org.rhodes.sipevent.pojo` Classes

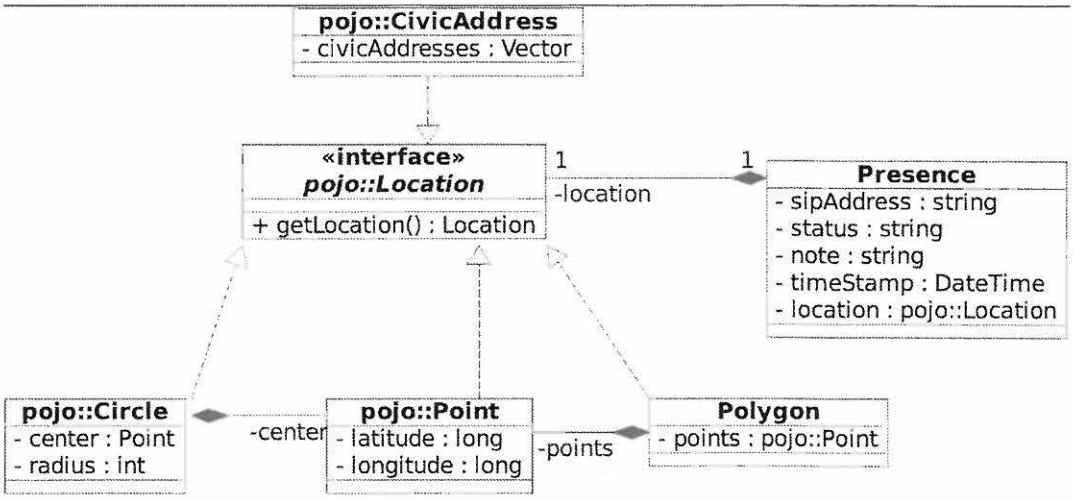

Figure A.2: Classes of the `org.rhodes.sipevent.pojo` package

We discussed the `org.rhodes.sipevent.pojo` package in Chapter 4. LBSs developers need to understand these classes and their relationships because instances of these classes hold the location data within the application. The `SipConnectionManager` methods for publication, subscription and notification either take in these instances or return them. This can be seen from the operation signatures in Figure A.1. The `Presence` class should have an instance of a class that implements the `Location` interface. These classes are the `CivicAddress`, `Circle`, `Point` and `Polygon` class. We implemented full functionality only for the use of the `CivicAddress` and `Point` class instances. Developers can refer to the *toolktitdemos* project to see an example of how the objects of this package are handled during notifications and how they can be used for location and presence publications.

As far as the transport of location and presence information using the JME LBSs SDK is concerned, the guidelines discussed here should be sufficient to assist the developer. We will discuss the guidelines concerned with the transport of location and presence information using the modified MPS in section A.3.2.1. Before that, we discuss the guidelines that should be followed when the JME LBSs SDK is used to add privacy handling to LBSs applications.

A.2 Guidelines for Building Basic Location-Based Services Using the Location-Based Services Development Kit: Information Privacy

In addition to the SIP presence and location transport facilities, the JME LBSs SDK also facilitates the development of location privacy tools. This section presents the general guidelines for using the privacy facilities of the SDK in developing LBSs. We emphasise what the SDK can perform on behalf of the developer.

A.2.1 The GeolocationPolicy and GeolocPolicyAgent Classes

The `GeolocationPolicy` class has one sole purpose: to allow LBSs developers to manipulate geolocation-policy documents without knowledge of the XCAP protocol. Each mobile device application must run only one `GeolocationPolicy` instance at a time. Developers should extend the `GeolocPolicyAgent` abstract class, discussed in the previous chapter, in all applications that manipulate geolocation-policy documents. This class creates and instantiates a `GeolocationPolicy` object that could be used in LBSs applications. The abstract class downloads the geolocation-policy document of a given Target if one exists at the XDMS. Otherwise, if no document exists, the developer has to put together a default document to be used by the `GeolocationPolicy` instance. If the user does not have an account at the XDMS or the username and passwords for the Target are wrong, the `GeolocPolicyAgent` instance informs the developer of this. Other operations provided by the `GeolocPolicyAgent` include adding and removing watchers from given geolocation-policy document rules. This is a common operation in many privacy-aware LBSs, therefore warranting its inclusion in the `GeolocPolicyAgent` class. The class should be imported using the line `import org.rhodes.LBSs.GeolocPolicyAgent;`.

A.2.1.1 Guidelines for Creating a Default Geolocation-Policy Document

The `GeolocPolicyAgent` abstract class has a default geolocation-policy document, shown in Listing A.1, that has two basic rules namely `closeties` and `looseties`. This document is put to the XDMS if a developer does not provide his or her own default geolocation-policy document. As mentioned in the preceding chapter, the two rules are different only in

what they provide within their `transformations` elements. The `closeties` rule provides no privacy filtering and the location information of a Target is provided to the LCs as it is. The `looseties` rule provides location-based privacy filtering with a transformational radius of 5 kilometers if the published location information of the Target is geodetic (please refer to Chapter 5 for location-based privacy filtering). If the location information of the Target is published in civic format, it is reduced to `city` level (i.e level of the `A2` civic address element).

Developers who provide their own default geolocation-policy documents must pass them to the constructor of the parent `GeolocPolicyAgent` class of their application. The format of the document should follow the geolocation-policy document schema specified in [66].

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <cp:ruleset xmlns:gp="urn:ietf:params:xml:ns:geolocation-policy"
4  xmlns:cp="urn:ietf:params:xml:ns:common-policy"
5  xmlns:ca="urn:ietf:params:xml:ns:pidf:geopriv10:civicAddrr"
6  xmlns:gml="http://www.opengis.net/gml"
7  xmlns:lp="urn:ietf:params:xml:ns:basic-location-profiles"
8  xmlns:gs="urn:ietf:params:xml:ns:pidf:geopriv10:geoShape">
9
10 <!-- first rule starts here -->
11
12 <cp:rule id="closeties">
13  <cp:conditions>
14   <cp:identity></cp:identity>
15   <gp:location-condition></gp:location-condition>
16  </cp:conditions>
17  <cp:actions>
18   <gp:sub-handling>allow</gp:sub-handling>
19  </cp:actions>
20  <cp:transformations>
21   <gp:provide-location/>
22  </cp:transformations>
23 </cp:rule>
24
25 <!-- second rule starts here -->
26
```

```
27 <cp:rule id="looseties">
28  <cp:conditions>
29   <cp:identity></cp:identity>
30   <gp:location-condition></gp:location-condition>
31  </cp:conditions>
32  <cp:actions>
33   <gp:sub-handling>allow</gp:sub-handling>
34  </cp:actions>
35  <cp:transformations>
36   <gp:provide-location profile="civic-transformation">
37    <lp:provide-civic>city</lp:provide-civic>
38   </gp:provide-location>
39   <gp:provide-location profile="geodetic-transformation">
40    <lp:provide-geo radius="5"/>
41   </gp:provide-location>
42  </cp:transformations>
43 </cp:rule>
44
45 </cp:ruleset>
```

Listing A.1: The default geolocation-policy document of the `GeolocPolicyAgent` class

A.2.1.2 Modifying Geolocation-Policy Document Rules

To allow for the customisation of the rules by users of the LBSs, it is useful that LBSs developers provide the means to change the default geolocation-policy document. Figure A.3 shows the `GeolocationPolicy` class and its public attributes and methods that can be used for geolocation-policy document operations. Location conditions can be added by invoking the `addGeodeticConditionToRule` and `addCivicConditionToRule` methods of the GeolocationPolicy class. All location conditions must have a unique `conditioId` which is taken in as a parameter to these two methods to identify the condition. Geodetic conditions take in instances of the Circle class from the org.rhodes.sipevent.pojo package. The `addCivicConditionToRule` method takes in instances of the `CivicAddress` class as parameters. The `putGeodeticTransformation` and `putCivicTransformation` methods allow for geodetic and civic transformations to be added to specified rules, respectively. For information regarding the rest of the `GeolocationPolicy` class methods, please consult

the Javadoc provided with this thesis.

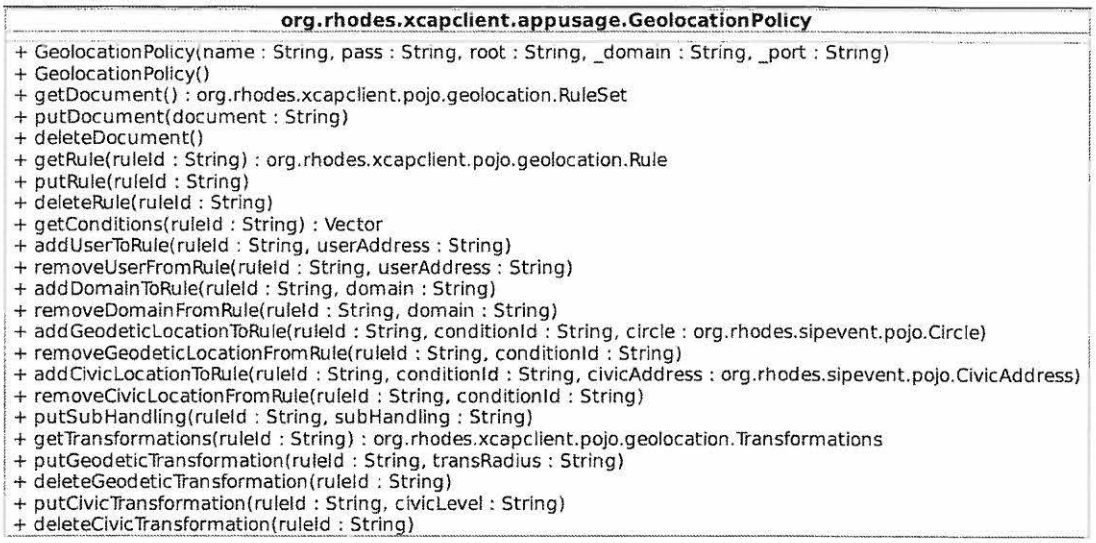

Figure A.3: The GeolocationPolicy class and its public attributes and methods

A.2.1.3 Adding Users to Geolocation-Policy Document Rules

```
1 public void addFriend(String watcher, String rule) {
2       super.addWatcherToRule(watcher, rule);
3       sipConnectionManager.addSubscriptionTask(watcher);
4 }
```

Listing A.2: The *addFriend* method of the FriendFinder class

The `addWatcherToRule` and `removeWatcherFromRule` methods of the `GeolocPolicyAgent` class can be used for adding and removing watchers to and from given geolocation-policy document rules. Listing A.2 shows the `addFriend` method of the `FriendFinder` class (from the *toolktitdemos* project) that invokes the `addWatcherToRule` method of the parent `GeolocPolicyAgent`. The method provides the SIP address of the watcher to be added and the name of the rule to which they should be added. At line 3, a subscription dialog with the watcher is started by invoking the `addSubscriptionTask` method. In a real-life scenario, a developer would need to check if the `addWatcherToRule` method executed successfully before the `addSubscriptionTask` method is invoked at line 3. This can be done by implementing the methods of the `NotificationListener` interface or by invoking

the `getResponse` method of the `GeolocationPolicy` class. We have not done it here, however, to keep our demonstration simple.

A.2.1.4 The `org.rhodes.xcapclient.pojo` Classes

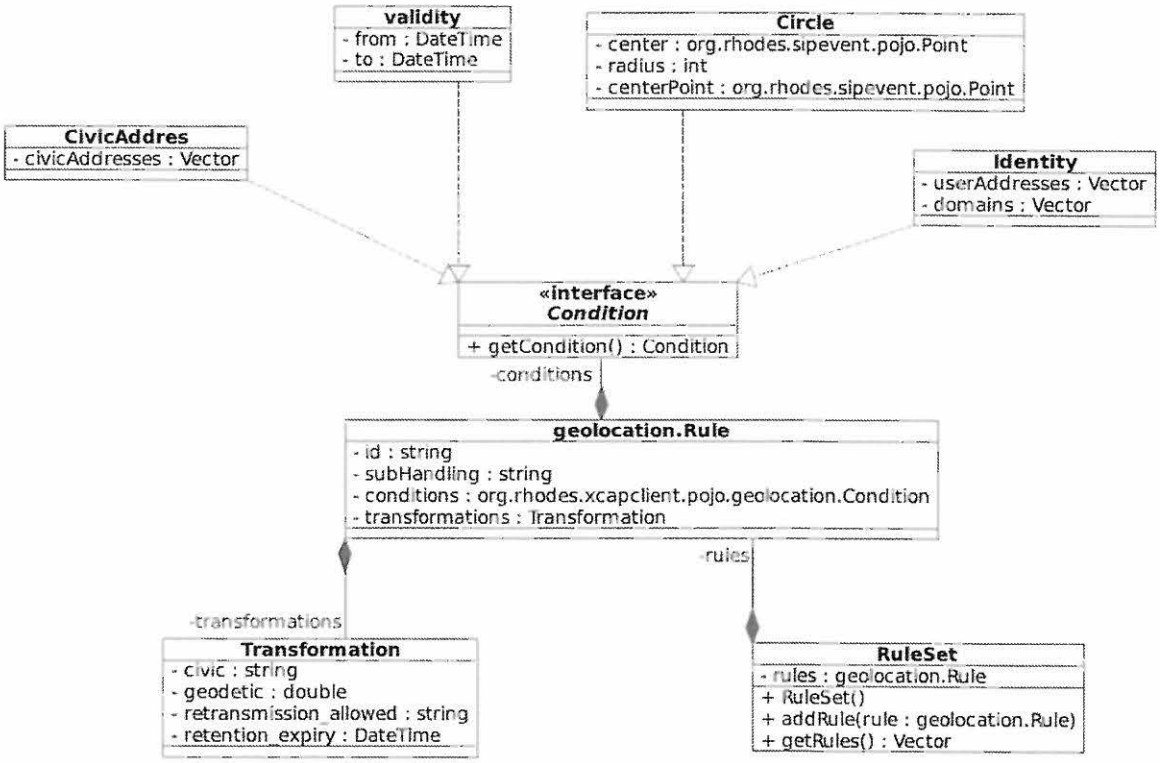

Figure A.4: Classes of the `org.rhodes.xcapclient.pojo` package

The methods of the `GeolocationPolicy` class that get XCAP resources from the Mobicents XDMS acquire the whole or portions of the geolocation-policy document. The `GeolocationPolicy` class parses the information from the geolocation-policy document and return them using instances of the classes from the `org.rhodes.xcapclient.pojo.geolocation` package. This can be seen from the signatures of the operations shown in Figure A.3. Therefore, for LBSs developers to be able to access the data properly, they need to have a general understanding of the classes in the package.

As it can be seen in Figure A.4, a `RuleSet` class is an aggregation of `Rule` objects. The `getDocument` method of the `GeolocationPolicy` class returns a `RuleSet` object contain-

ing all the geolocation-policy rules of the Target. Every `Rule` instance can have one or more `Condition` interface instances. These instances could be from the `Identity`, `Validity`, `Circle` and `CivicLocation` classes that can hold the data for different types of conditions. Here the developer can make use of the `instanceof` operator to probe for the type of `Condition` instance. It is also important to note that some methods get portions of the geolocation-policy document. For instance, the `getRule` method of the `GeolocationPolicy` class returns a `Rule` object and the `getCondtions` method returns a collection of `Condition` objects.

A.3 Setting up and Using the Modified Mobicents Presence Service and XML Document Management Server

The MPS originally managed basic presence and other presence extensions (the IETF Presence Data Model and the RPID). The Mobicents XDMS provided the presence-rules application usage (appusages) to provide privacy protection services to the MPS. In Chapters 4 and 5, we discussed how we modified the MPS so that it processes only basic presence and location information. We added the geolocation-policy application usage (appusage) to the Mobicents XDMS to provide location privacy to the modified MPS. This was discussed in Chapter 5.

Service developers are expected to put together their applications using the JME LBSs SDK without any coding or modifications at the modified MPS or XDMS. For this reason, developers need only to have an overview of the services provided and System administrators need to know how to set up these servers. These two points are discussed in the next two sections.

A.3.1 Systems Administrator: Setting Up the Modified Mobicents Presence Service and XML Document Management Server

A.3.1.1 Setting Up the Modified Mobicents Presence Service

The modified MPS is set-up in exactly the same way as the original MPS since only the inner information processing was changed. The setup instructions are as found at http://docs.jboss.org/mobicents/sip-presence/1.0.0.CR1/user-guide/en-US/html_single. It is particularly important to note from the setup instructions that the MPS depends on the SIP11 resource adapter. The reader is advised to use the SIP11 resource adepter provided with this thesis as it contains the modifications mentioned in section 4.2.2.1.

Our modifications were performed on Beta 5 version of the MPS. The MPS runs in a JBoss application server container, so administrators are required to set up the JBoss application server first. The modified MPS was tested on version 4 (specifically version 4.2.3.GA) of the JBoss server. The JBoss server folder provided with this document contains both the MPS and XDMS modified as part of our work as well as their dependencies. Administrators must set up this server to use the modified MPS and XDMS hosted within it. Administrators must note the following points:

- The MPS listens for connections on the SIP port (5060) and the XDMS on the HTTP port (8080).
- The JBoss server and MPS provided in this project depend on version 5 of the Java Development Kit (JDK) and the version 1.5 Java Run-time Environment (JRE) because these were the current versions when the project began.

The provided JBoss server (containing the modified MPS and XDMS) can be set up by copying the given folder in a local home drive and configuring the environment variables of the server using instructions provided in section 2.1.3 of the MPS setup guide at http://docs.jboss.org/mobicents/sip-presence/1.0.0.CR1/user-guide/en-US/html_single. Once installed, the JBoss server can be run by executing the run.sh script on Unix or run.bat script on Microsoft Windows. These scripts are contained within the <j_boss_install_directory> /bin directory on Unix or Windows. The services can be shutdown by executing the shutdown.sh script on Unix or shutdown.bat script on Microsoft Windows.

A.3.1.2 Setting Up the Modified Mobicents XML Document Management Server

We discussed the process of setting up the JBoss server that contains the modified MPS and Mobicents XDMS in section A.3.1.1. Setting up and running the JBoss server provided with this document is all that is needed to run the Mobicents XDMS. The setup instructions are also found at http://docs.jboss.org/mobicents/sip-presence/1.0.0.CR1/user-guide/en-US/html_single. The reader should take note that the XDMS depends on the HTTP Servlet Resource Adapter. The XCAP interface which is used to manage geolocation-policy information on the Mobicents XDMS is public. Users must create profiles to enable information protection and XCAP request authentication. Profiles can be added or removed from the XDMS by using the JBoss Management Bean given below:

`org.mobicents.sippresence:name=UserProfileControl`

Figure A.5 shows the interface provided by this JBoss Management Bean. The user puts the SIP address of a Target as the first parameter of the `addUser` operation. Passwords are provided as the second parameter.

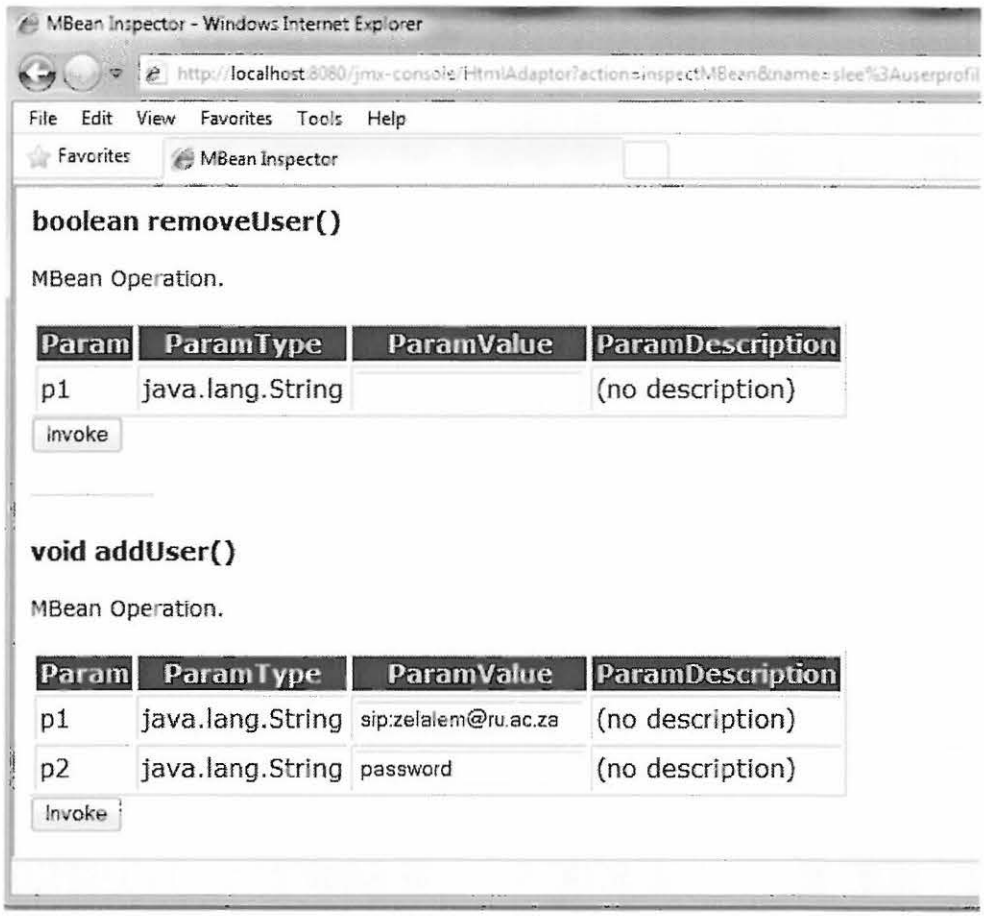

Figure A.5: The Mobicents XDMS web interface for creating and removing XDMS profiles

A.3.2 Service Developer: An Overview of the Services Provided By the Mobicents Presence Service and XML Document Management Server

A.3.2.1 Services Provided by the Modified Mobicents Presence Service

The modified MPS is composed of SBBs that manage basic SIP presence and location information publications. The location information is contained with Location Objects of PIDF-LO documents. The MPS also has SBBs that manage SIP subscriptions to the Targets that have published their basic presence and location information. After setting

up the modified MPS as part of the JBoss server discussed in the preceding section, the above-mentioned services are available for use.

When a publication is made by a Target, the modified MPS replaces the old presence and location information with the newly published information. The modified MPS can receive only points, circles and polygons and supports only the levels of civic information discussed in 4.5.1.1. The limitation, however, is that the subscription SBBs are programmed to offer location privacy filtering only to geodetic points and only six levels of civic information as discussed in 5.4.3.2.

The modified MPS offers a limited location notification filtering service. This service enables SIP notifications to be made to subscribers when Targets are within a given geoshape circle. The geoshape circle used for notification filtering is provided to the modified MPS only through the first SIP subscription request made to the MPS. The notification filtering service is limited by the fact that only geoshape circles can be used in notification filtering documents. Also only the enterOrExit filtering mechanism specified in [78] can be used.

The MPS can be used with other SIP clients that publish or subscribe to location information as long as they conform with the presence IETF RFC given in [14].

A.3.2.2 Services Provided by the Mobicents Mobicents XML Document Management Server

The original Mobicents XDMS supports the presence-rules and resource-lists appusages. We added the geolocation-policy appusage to provide location privacy to the modified MPS. The server uses digest authentication to provide private access to the geolocation-policy documents belonging to different Targets. The JME LBSs SDK is designed to work with this XDMS, and to negotiate the authentication parameters appropriately. Developers provide a password and username when they extend the GeolocPolicyAgent abstract class. Service users need to create profiles at the Mobicents XDMS as discussed in section A.3.1.2. Developers can invoke XCAP GET, PUT and DELETE operations on geolocation-policy documents by invoking methods of GeolocationPolicy instances. The default XCAP root of the server is *mobicents* and the default port is 8080.

The geolocation-policy appusage that we added to the XDMS can work with any XCAP client that conforms to the standards specified at the geolocation-policy RFC draft specified in [66].

Geolocation-policy documents are used during SIP subscriptions. The location-aware MPS fetches from the XDMS the documents of all the Targets whose location information is being subscribed to. The contents of the document are parsed and used to authorise subscriptions and to control the granularity of the location information that is shared with the LCs (location privacy filtering discussed in Chapter 5).

A.4 Summary

Chapters 4, 5 and 6 focused on the development of the JME LBSs SDK and on modifying the MPS and XDMS. Developers need no knowledge of SIP or XCAP protocols to develop services with the tools we have put together in this project. This appendix discusses how the LBSs tools should be used by developers to put together their LBSs. The JME LBSs SDK facilities either support the transport of presence and location information (using the modified MPS) or they communicate the privacy preferences of Targets to the Mobicent XDMS.

Firstly, we discussed the setting up of the transport facilities of the JME LBSs SDK. The `SipConnectionManager` class is the class that developers must use to perform SIP location information registrations, publications and subscriptions. We highlight its functionality, the services it provides to developers, and general guidelines regarding its usage. Secondly, the privacy facilities of the JME LBSs SDK were discussed with primary focus on the `GeolocationPolicy` and `GeolocPolicyAgent` classes. These classes can be used to perform geolocation-policy document operations without knowledge of the XCAP protocol. The classes and interfaces provided in the org.rhodes.LBSs, the org.rhodes.sipevent.pojo and the `org.rhodes.xcapclient.pojo.geolocation` packages are important for service development and we discussed them here as well.

At this point, we expect developers to build their services using the JME LBSs SDK. For this reason, the discussion of the server-side infrastructure we provided here focused on setting the servers up (for systems administrators) and highlighting the services that they provide (for developers).

Appendix B

Publications Resulting From This Research Effort

1. S. Ndakunda, A. Terzoli, M. Wright. An Investigation of Location-Based Services in the Mobicents Framework. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, 2009.

2. S. Ndakunda, A. Terzoli, M. Wright. Adding Location-Based Services Support to the Mobicents Service Delivery Platform. In *Southern Africa Telecommunication Networks and Ap- plications Conference (SATNAC)*, 2010.

3. S. Ndakunda, Z. Shibesi, K. Bradshaw. Delivering a Personalised Video Service Using IPTV. In *International Conference on Advanced Communication Technology (ICACT)*, 2011.

4. S. Ndakunda, A. Terzoli, M. Wright. Composing a Simple Friend-finder Application Using the SIP Location-based Services Toolkit. In *Southern Africa Telecommunication Networks and Ap- plications Conference (SATNAC)*, 2010.