A Presence Server for Context-aware Applications

MOHAMMAD ZARIFI ESLAMI

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2007

COS/CCS 2007-27

A Presence Server for Context-aware Applications

Mohammad Zarifi Eslami
moze@kth.se

Department of Communication Systems
School of Information and Communication Technology
Royal Institute of Technology (KTH)
Stockholm, Sweden

17 December 2007

Supervisor & Examiner: Professor Gerald Q. Maguire Jr., PhD

Technical Advisor: Athanasios Karapantelakis

To my wife and my parents.

Abstract

This master's thesis project "A Presence Server for Context-aware Applications" was carried out at KTH Center for Wireless Systems (Wireless@KTH). The overall goal of this thesis project is to implement a context aware infrastructure to serve as middleware for different kinds of context aware applications, such as a context-aware printing application, location based notifier application, etc. This thesis examines different types of context aware architectures and considers different forms of context modeling. Additionally the thesis also explores some of the related technology, in order to provide the reader with suitable background information to understand the rest of the thesis. By using the SIP Express Router (SER) and its presence module (pa) a context server has been designed, implemented, and evaluated. Evaluation reveals that the critical bottleneck is the increasing service time as the number of Publish messages for different events in the SER database increases, i.e. the time required for handling and sending the Notify messages when a new Publish message is received increases as a function of the number of earlier Publish messages. The evaluation also shows that the dependence of SER upon the MySQL database as incorrect database queries can cause SER to crash. Additionally the performance of the database limits the performance of the context server. A number of future improvements are necessary to address security issues (in particular the authentication of Watchers) and adding policy based control in order to send Notify messages only to the Watchers authorized *to receive information for a specific event.*

Sammanfattning

Examensarbetet "A Presence Server for Context-aware Applications" genomfördes på Kungliga Tekniska Högskolan, KTH Center for Wireless Systems (Wireless@KTH). Det övergripande målet med detta examensarbetsprojekt är att implementera en kontextmedveten infrastruktur som fungerar som "middleware" för olika typer av kontextmedvetna applikationer. Exempel på dessa är kontextmedveten utskriftsapplikation och platsberoende meddelarapplikation osv. Rapporten undersöker olika typer av kontextmedvetna arkitekturer och betraktar olika former av kontextmodellering. Rapporten utforskar även vissa besläktade teknologier för att kunna tillhandahålla läsaren med en passande bakgrundsinformation och därmed öka förståelsen för resten av examensarbetet. Genom att använda Sip Express Routern (SER) och dess närvaromodul (presence module, PA) har en kontextserver designats, implementerats och utvärderats. Utvärderingen visar att den kritiska flaskhalsen är tiden det tar för SER servern att svara på nya Publish meddelanden, för olika händelser, i SER databasen. Svarstiden ökar allteftersom databasen fylls med mer data. Detta påverkar hantering och sänding av Notify meddelande när en ny Publish meddelande är mottagen. Uvärderingen visar också att en viktig fråga är relationen mellan SER servern och MySQL databasen, eftersom felaktiga förfrågningar till databasen kan krascha SER servern. De viktigaste framtida förbättringarna är säkerhetsaspekter (mer specifikt autenticering av Watchers) och tillägg av policybaserad sändning av Notify meddelanden endast till auktoriserade Watchers för specifika händelser.

Acknowledgement

This thesis project has taken placed at the Computer Communication Systems laboratory of Wireless@KTH between June and December 2007. Undoubtedly, during this period, I have experienced one of the most effective times in my life. The warm and friendly environment at Wireless@KTH was truly motivating and I am proud to have been given this chance to experience research in such an environment.

Before anyone, I would like to express my most sincere gratitude to Professor Gerald Maguire, who accepted me as his master's thesis student and supported me to create one of the most wonderful periods of my educational life. Remembering his nice and friendly behavior always accompanied with a smile was a big encouragement for me to overcome the problems encountered during my work. His way of analyzing my results were the most instructive lessons and provided me with excellent suggestions to improve the quality of my work. Those favors will never be forgotten! Never.

I am also deeply indebted to my very resourceful technical advisor, Athanasios Karapantelakis, whom helped me with some of the technical parts of the thesis project. His ability to simplify complex problems and his great programming knowledge helped to teach me how to manage and plan in order to deal with the problems in my life logically! In my opinion, he is an outstanding individual with great abilities and will always be in my memories and thoughts. Thanks for all your instructions!

Contents

List of Figures

List of Tables

1 Introduction

Humans continually try to find new ways to make life easier. Individuals invent new machines to work on their behalf and try to automate their jobs. In order to create a more intelligent device (i.e., to reduce the effort necessary for an individual to use this device to accomplish the task which they desire), the device should acquire information concerning the user and user's context, then process it in order to decide what function the device should perform. In a specific application, this relevant information concerning users is called context. Context information is any information that can characterize a user and his/her current situation, such as: location (home, office, car, conference room, on vacation, etc.), activity (working, resting, having lunch, talking with their boss, etc.), time (working hours, weekend, etc.), user preferences (accepting calls from family, rejecting call from friends, etc.), … [5]. A context aware system is an intelligent system which can react automatically based on the user's current context information on behalf of this user. As an example, a context aware printing system finds available printers on a university campus and recommends the appropriate printer to the user for the user's current printing task. This application will utilize context information such as user location, user preferences, document type, printer's toner/ink/… status, etc. in order to recommend one or more printers as the most appropriate printer(s) to the user. Similarly a context aware server can decide on behalf of a user whether to accept/reject/redirect an incoming/outgoing call based on this user's current context and previously specified preferences.

The Wireless@KTH center was chosen as the environment for a pilot test in order to evaluate some context services with the proposed middleware context architecture (presence server). As this center exists within a university campus, related raw context information will be collected from students, teachers, and visitors as they are the users and this is their environment. After processing this raw data, the results will be distributed to different applications; each of these applications will then attempt to simplify the user's life (or at least reduce the amount of interactions which they require to perform the task which they are most interested in performing) or improve the quality of life for these users (which again is often related to simplifying the number of choices which the user is called upon to make).

In this master thesis, I used the combination of a SIP Express Router (SER) server, its presence agent (PA) module, and MySQL database to implement my desired presence server. This presence server works as a context server for different type of context-aware applications in order to : (a) obtains the updated context information, (b) reads, processes, and stores this context information in the local MySQL database SER, (c) notifies only interested Watchers about this context information. SIP for Instant messaging and presence leveraging extensions (SIP-SIMPLE) protocol (an extension of SIP to support instant messaging and presence) is used to distribute context information among entities and Presence Information Data Format (PIDF) is used as a context model to transfer this context information in a standard format.

Different types of technologies are required for implementing a context infrastructure. Each of these technologies will be addresses in the subsequent parts of this thesis. This breadth of technologies is clearly reflected in the organization of this report. Chapter 2 is dedicated to a brief discussion of some of these related technologies, including: SIP, SER, XML terminology, and SIP-SIMPLE. Chapter 3 describes a context aware infrastructure in detail; accompanied by context modeling and discovery and a brief review of related work is also presented at this chapter. Chapter 4 is about the main goals of this master thesis, and Chapter 5 describes the implementation of a prototype of a proposed infrastructure, while chapter 6 presents an evaluation of this infrastructure. Finally chapter 7 presents some conclusions and suggestions for future work.

2 Background

2.1.1 SIP

The Session Initiation Protocol (SIP) [16] is strongly associated with IP telephony, but because of some SIP features (see later in this section) there are additionally areas of use. Generally signaling protocols which are used to set up and terminate calls, carry information required for each call (such as media CODEC (a technique used to compress/decompress speech or audio signals), IP addresses, and port numbers), locate users to be called, negotiate capabilities, and invoke services such as hold, mute, and transfer. There are four major protocols currently used for signaling IP telephony services: H.323, the Media Gateway Control Protocol (MGCP) [71], the Session Initiation Protocol (SIP) [19], and Skype [72] (which we will not consider further since it is a proprietary protocol). Here we will focus on SIP as it is a widely used protocol which is implemented by a number of open source software packages and is desired to be highly extensible.

SIP is a text-based protocol, similar to HTTP for initiating interactive communication sessions between users. It is used for creating, modifying, and terminating sessions with one or more participants. Sessions include: voice, video, chat, interactive games, virtual reality [7], and other interactive media sessions. SIP was initially defined in 1999 in RFC 2543 [15] and has evolved to the version which described in RFC 3261 [16].

As be shown in the Figure 1, if Alice wants to make a phone call to Bob, first Alice may send a SIP Invite message to her outbound SIP proxy server, this proxy utilizes a Domain Name System (DNS) server to find's Bob Inbound proxy server, then this proxy learns Bob's location by asking its location server; if this proxy learns Bob's location it answers with an OK message, to which the outbound SIP proxy responds with an ACK message (assuming that the available set of CODECs and other details match). At this point a session between Alice and Bob has been created and they can directly communicate in this session using a media protocol such as the Real-time Transport Protocol (RTP). RTP defines a standardized packet format for delivering audio and video over the Internet [8]). Note that RTP operates directly without requiring participation of the SIP proxies, until the parties want to terminate the session; session termination is done by sending SIP BYE messages. Thus in contrast with traditional telephony where everything (signaling and media) is handled by the network; in the case of Voice over IP (VoIP), the network is responsible only for signaling, while the media packets are transmitted directly between caller and receiver.

As we can see in the following figure, the SIP protocol depends upon request-response messages, which form a "SIP Transaction". Additionally we describe (below) several different types of requests, which are part of SIP.

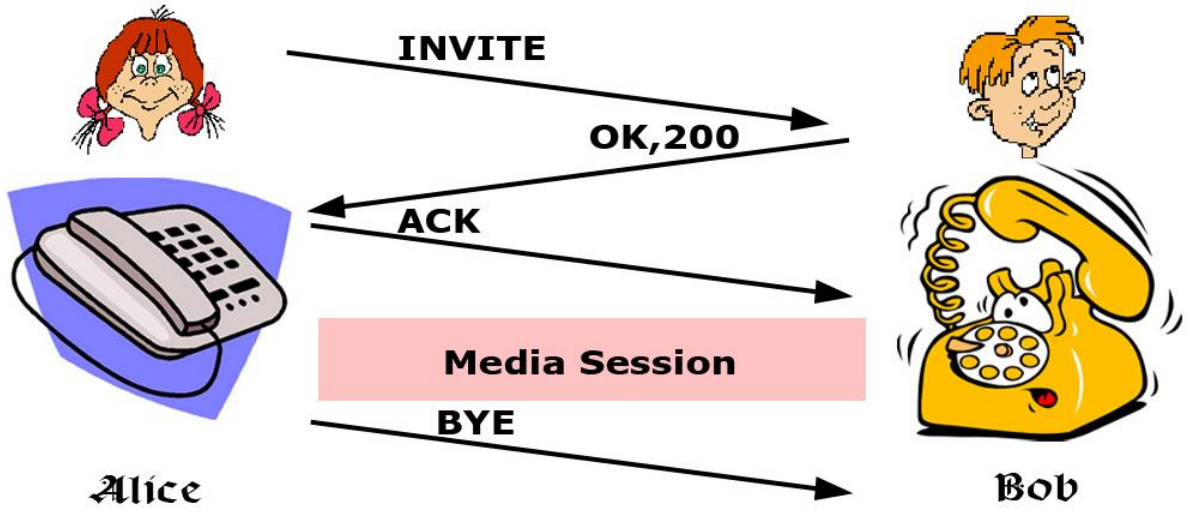

Figure 1. SIP messages, involved when Alice invites Bob to a SIP session. Later Alice terminates this session.

INVITE	Invite a user (Bob) to participate in a call session (Alice can initiate a SIP communication with Bob by sending an INVITE request to the server).
REGISTER	Registers the address listed in the header field with a SIP server.
ACK	Confirms that Bob has received a final response to an INVITE request.
OPTIONS	Queries the capabilities of servers.
CANCEL	Cancels any pending searches but does not terminate an already accepted call.
BYE	Terminates a call and can be sent by either the caller (Alice) or the callee (Bob).

A three digit integer with a short description will be sent as a response to a request:

PROVISIONAL (1xx)	Request received, continuing to process the request.
SUCCESS (2xx)	The action was successfully received, understood, and accepted.
REDIRECTION (3xx)	More action needs to be taken in order to complete the request.
CLIENT_ERROR (4xx)	The request contains bad syntax or cannot be fulfilled.
SERVER_ERROR (5xx)	The server failed to fulfill an apparently valid request.
GLOBAL_ERROR (6xx)	The request cannot be fulfilled at any server.

Hypertext Transfer Protocol (HTTP) is an IP protocol used for hypertext transfer; while SIP is a signaling protocol designed for establishing sessions. In the case of SIP a user has a

globally reachable address, such as: *SIP:Mohammad@kth.se.* SIP provides a mechanism for mapping this Universal Resource Identifier (URI) to the user's current device, thus providing user mobility. A SIP URI must include a component which can be resolved to a host name or address, but may include a user name, port number, etc. Note the recipient of a call determines if the caller is able to see any of the user's addresses other than their SIP URI.

The SIP architecture specifies five important entities:

User Agent (also called a SIP endpoint): functions as a client when initiating requests and as a server when responding to requests. Each user agent has a SIP URI: this could identify a SIP phone, a PDA, laptop, etc. [7].

SIP Proxy Server: forwards requests from a User Agent to the next SIP server (i.e., receives SIP requests and forwards them on behalf of the requester).

Location server: interact with the SIP proxy server to return the calle's current location. Note that this location is not necessarily revealed to the caller.

SIP Redirect Server: responds to client requests with the requested server's address or redirects the caller to another server or servers.

Registrar: accepts registration information from a SIP User Agent, and stores the location of this user agent using it in a location service (via a non-SIP protocol) [15].

All of these elements are logical entities and may be implemented as a single server.

2.1.2 Why we use SIP?

There are a couple of reasons which we have chosen to utilize a SIP infrastructure for this thesis project. First SIP is powerful and at the same time easy to implement. Furthermore SIP is scalable and open standard. Hence there are numerous freely available SIP implementations; many of which have a both large developer and user communities. Many of these implementations also support different applications; in particular one of these provides additional functionality to support the Call Processing Language (see section 2.5).

SIP is a text-based protocol, so its messages are human-readable, which makes debugging, reading logs, and finding errors easier. Additionally, SIP supports mobility and provides Instant Messaging and Presence (IMP) services for mobile devices.

2.2 XML

The Extensible Markup Language (XML) is a general-purpose markup language used to facilitate sharing data across different information systems, generally via the Internet [10]. XML is a strict language in comparison with HTML. In HTML, parsing rules are forgiving. Unknown tags may be silently ignored, while missing required tags may be added. In HTML, some tags do not need to be closed, while in XML every opened tag must be closed.

XML was developed by the XML Working Group (originally known as the Standard Generalized Markup Language (SGML) Editorial Review Board) and recommended by the World Wide Web Consortium (W3C) in 1996 [12]. It is a fee-free open standard, with the following features:

- Designed to make it easy to interchange structured documents over the Internet.
- A simplified subset of the Standard Generalized Markup Language (SGML).
- Supports a wide variety of applications, can be used on a wide variety of platforms, and interpreted with a wide variety of tools.
- Flexible enough to be able to describe any logical text structure.
- Documents are readable both by humans and machines, as well as being easy to create.
- More than a markup language as it allows you to describe languages.
- By defining the role of each element in a formal model, known as a Document Type Definition (DTD) or by using an XML schema, users can effectively create "extensible" tag sets that can be used for multiple applications (Extensibility), unlike Hypertext Markup Language (HTML).
- XML has a strict syntax, for instance, each opened tag <> should be closed </>.

XML was not designed to do anything itself, rather XML was created to structure, store, and to encode information [14]. The following example is the location information (context) of a user encoded as XML, which later used in this thesis:

<**location**>

<**description**>Wireless</ **description** >

<**floor**>2</**floor**>

<**room**>Grimeton</**room**>

</**location**>

As can easily be seen a **location** has a description, floor, and room elements; however this XML simply provides pure information wrapped in XML tags. Someone must write software to send, receive, or display this information, but XML makes it easier for the receiver to know what the information is [14]. However, XML itself does not provide any meaning to the tags.

Due to the features explained above, context modeling information can be represented as an XML document (context modeling is used to easily store, process, and deliver context). In a number of different applications, such as the printing context application (see section 3.2) various elements of context information (tags) can be easily defined in the XML DTD file or by using an XML schema (which defines each element of different types of contexts (for more information about XML DTD or Schema refer to web page [58])).

2.3 SER

SIP Express Router (SER) is a high-performance, configurable, open source SIP server [24]. It was developed by a team employed by Fraunhofer Fokus (a German research institute) in 2001. In 2004, part of the Fokus team moved, with the SER copyrights, to a newly created company, iptel.org [17] that worked on Voice over IP services, and was lead by Jiri Kuthan. SER was offered as open source (GNU Public License, GPL) and the iptel.org website is still the main entry point for further SER information, as well as SIP tutorials and other related resources [23]. Two of the SER core developers and one main contributor started a new Open Source project called OpenSER in 2005. Note that OpenSER uses the same SER configuration files (ser.cfg) as SER [22]. The "ser.cfg" configuration files control which modules should be loaded and define how the modules should behave by setting module variables. An example of such a configuration file can be found in appendix A.

A SER server can receive and process SIP messages to enable intelligent VoIP Services [4]. We use SER to register users in a database (which acts both as a general database and as the SIP location server) enabling SIP messages to be routed between clients, service agents, applications, and sensors. We explain how to install and configure a SER for a Linux environment in Appendix A. Appendix A also describes how to install and configure the MySQL (database) server for SER.

2.4 SIP-SIMPLE

An instant messaging and presence system allows users to be notified of changes in user presence by subscribing to notifications for changes in state. There are alternative protocols to consolidate a standard method for instant messaging [49]: IETF's SIP-SIMPLE, APEX (Application Exchange), Prim (Presence and Instant Messaging Protocol), the open XML-based XMPP (Extensible Messaging and Presence Protocol), more commonly known as Jabber, and OMA's (Open Mobile Alliance) IMPS (Instant Messaging and Presence Service) created specifically for mobile devices.

SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) is an extension of SIP to support instant messaging and presence (IMP) functionalities. It provides a method for users to subscribe to events using other user's SIP URIs, hence a user can be notified of changes in another user's state or notified of other events [27].

In spite of SIP-SIMPLE's name, it is not simple, but in comparison to the majority of others instant messaging and presence protocols it is relatively simple. SIMPLE is an open standard, which enables messages to be exchanged within a SIP session and provides a subscription based framework for an event notification.

As we can see in the Figure 2, the SIP-SIMPLE protocol has a number of different components [28]: A subscriber known as a watcher is a party which is interested in learning about updates to presence information, it sends a Subscribe request (each Subscribe request has a timeout and this subscription automatically expires at the specified time, unless it is renewed). A notifier (known as presentity) provides presence information to interested watchers. A Presence User Agent (PUA), provides presence information for a presentity (there may multiple PUAs for a given presentity). A Preference Agent (PA) is a logical entity which receives Subscribe messages and generates Notify messages for incoming Subscribe messages or changes in the preference state of a presentity.

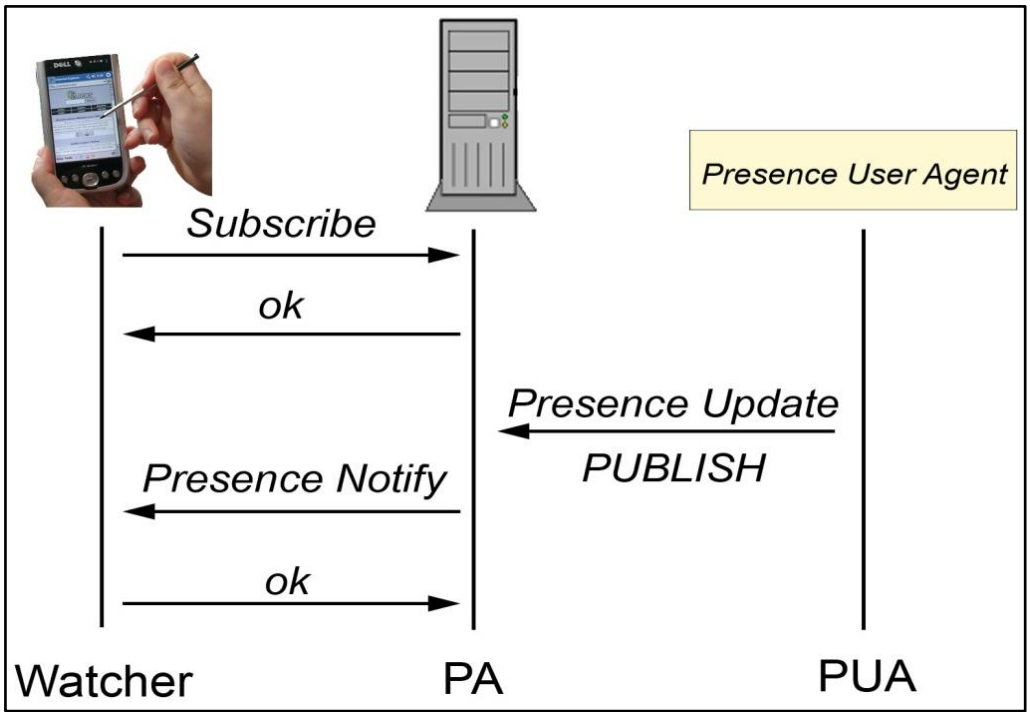

Figure 2. SIP-SIMPLE messages

In this thesis we will use SIP-SIMPLE, to distribute context to both users and applications. In addition to SIP-SIMPLE we can use SIP request-response mode (look at the Figure 3) to

allow every user to ask for a specific context, to which the service agent or centralized server (SER) will respond. Moreover by using SIP-SIMPLE method users do not need to ask for context, they simply subscribe to a specific service or application and whenever there is a change, the service agent will notify the subscribers. Depending upon the application we will use one of these two approaches. For example in the printing scenario it seemed to be better to use the SIP-SIMPLE method, where a user will subscribe to a specific printer once, then when there are changes in the status of this printer (from "ready" to "no paper") the user automatically will be notified that that printer is not available for printing and he/she should find another printer or in this case - should add paper to the input tray.

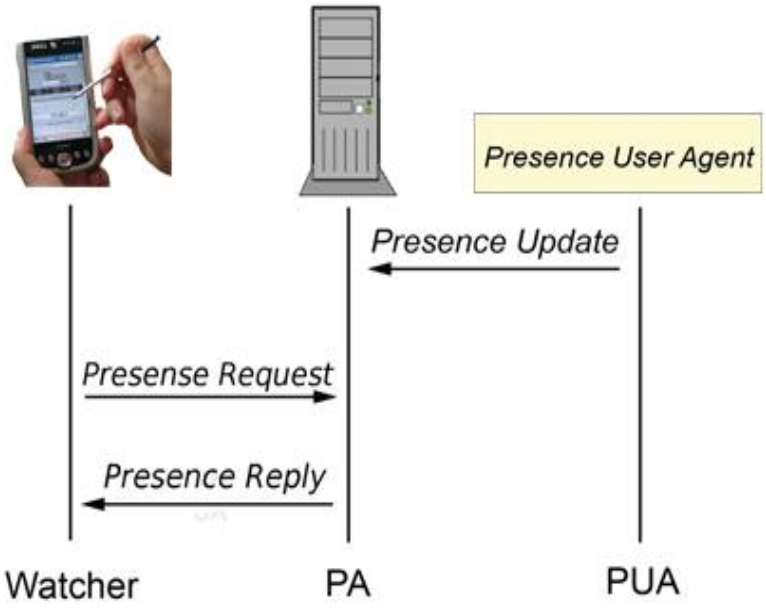

Figure 3. SIP Request/Response messages for obtaining presence updates

2.4.1 Publish, Subscribe, and Notify messages

In the below three different type of SIP-SIMPLE's messages is described in detail.

2.4.1.1 Publish

Publish is a SIP method that can publish the event state of a presentity [62]. The event state which the PUA wants to send to server is carried in the body of a Publish message, which in our case is encoded in Presence Information Data Format (PIDF, see section 3.4.3.1). In this thesis in order to inform the server about the changes of context, PUAs use Publish messages. Publish is similar to REGISTER as it can create, modify, and remove state information about another entity. The Publish method has a header (i.e. which includes information about the PUA and provides the server's address, message expiration time, etc.) and a body (i.e. which includes the context information). Whenever PUAs send a valid and

well-formed Publish message, the server replies with a 2xx message to indicate that it received the Publish request successfully. As described in RFC 3903 [62], There are 4 types of Publish messages:

Initial This is the first Publish message sent to the server. It contains the context information in its body. The server will generate and assign an entity-tag (SIP-ETag) and return this tag in the 2xx response to the PUA. The PUA must store this random value and use it in the other three messages. Note that this entity-tag establishes a conversation, which later should be explicitly closed (additionally the SIP-ETag provides mobility for PUAs. So if the device of a PUA changes, it can use this tag and keep its previous conversation with the server).

Refresh The refresh message does not have a body. The PUA uses the value of SIP-ETag (had been sent by the server in the initial message) in a SIP-If-Match tag, so that the server knows that this message belongs to the previous publication.

Modify If any changes occur in the event state, the PUA sends the updated context in the body of this message. Just as in the case of Refresh message, the Modify message has a SIP-If-Match tag.

Remove To inform the server that it should close the open conversation for a specific event, the PUA sends a Remove message which like the Refresh message does not have a body, but unlike the Refresh message this message has the value of '0' for the Expiration tag. After the server receives a Remove message for a given entity-tag, it removes all the state which it is maintaining about this tag.

Table 1. Four different types of PUBLISH messages

Message type	Body	SIP-If-Match	Expiration value
Initial	Yes	No	>0
Refresh	No	Yes	>0
Modify	Yes	Yes	>0
Remove	No	Yes	=0

Note that if the second Publish message does not have a SIP-ETag it will be consider to be a new Publish message (even though it uses the same Event package). To acquaint the reader with the different fields of a Publish message, Figure 10 shows how the different messages are used between PUAs, the server, and watchers. We will also examine the Publish messages used in our application.

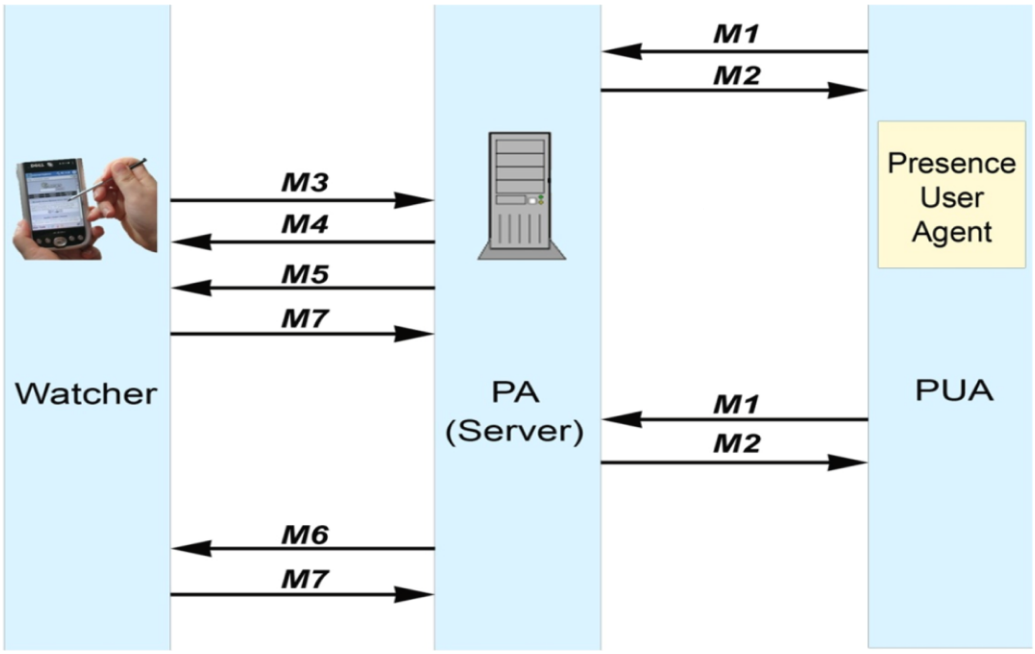

Figure 4. Messages flow between PUA, Server, and Watcher

The first publish message which our application uses, M1:

PUBLISH sip:ccsleft@130.237.15.238 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.196:5060;branch= z9hG4bKqeAQbxW
To: <sip:ccsleft@130.237.15.238>
From: <sip:ccsleft@130.237.15.238>;tag= qeAQ
Call-ID: 78@192.168.1.196
CSeq: 1 PUBLISH
Max-Forwards: 70
Expires: 2
Event: location
Content-Type: application/pidf+xml
Content-Length: 504
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
xmlns:location="http://it.kth.se/~moze/schemas/mohammad.xsd"entity="sip:ccsleft@130.237.15.238">
<tuple id="6sJ8J0">
<status><basic>open</basic>
<location>
<description>Electrum</description>
<room>Wireless</room>
<floor>1</floor>
<coordinates>
<latitude>123213</latitude>
<longtitude>47382145</longtitude>
</coordinates>
</location>
</status>
<note>location</note>
<contact priority=" 0.8">Mohammad</contact>
</tuple>
</presence>

In the first line, the word ‘PUBLISH’ indicates that this is a Publish message. There are two variables on this first line, the host name “ccsleft” and the IP address (or Domain name) of the server “130.237.15.238”. These variables can be configured to have the desired value, after SER has been installed (see section 5.2.1). The second line indicates that “UDP” is used as the transport protocol. There are three variables in the second line, the IP address of the PUA “192.168.1.196” and the port number “5060” (5060 is the default SIP port number that SER uses). The third variable ‘branch’, is a random number which should be different in different Publish messages. Note that based on RFC 3903, in the fourth and fifth line the ‘To’ and ‘From’ fields are **unlike** these fields in an Invite message; in the Publish message the ‘To’ and ‘From’ fields are same and both contain the **server’s** address and port number. Just as with the ‘branch’, the ‘tag’ is a random number which should be unique. In the next line ‘Call-ID’ is a field that has two parts, the first part needs to be random number, in this case the number “78”, and the second part is the IP address of the PUA. The ‘CSeq’ field contains a number which is incremented by one for every Publish message (the server replies to with an “OK” message for this Publish message using this same number), and “PUBLISH” again indicates that it is a Publish message. ‘Max-Forwards’ limit the number of hops a request can traverse to reach its destination. The ‘Expires’ value should set by the PUA to indicate how long this published update is valid (Note that in the ser.cfg file, the server can limit the maximum ‘Expires’ value, see Appendix E). In this Publish message the expiration time is “2” (which in our server, it means about 2 seconds, see chapter 6 for more information). Based on RFC 3903 and 3856, if a Publish message has expired, the server will not notify the PUA, but Watchers associated with this event should also be notified. However, in some applications (such as the context-aware printing application), there is no need to send the Notify message for the expiration of the Publish message. Therefore it is possible to disable this message by modifying the source code of the server (see section 5.2.2 and Appendix F), as a result the number of messages traversing the network will be reduced. (The reason for not sending this messages, is that the user or application is really only interested in the status of the printer at the time of their print job -- not at other times). The next line ‘Event’ field indicates that the PUA publishes the context information about a location event (our server can handle different event packages, and in our application we are using ‘location’, ‘presence’, and different place names as events). The server uses this ‘Event’ field to reduce the number of watchers which are notified; in this case it will notify only the interested subscribers in this event. If a PUA uses an event which is not defined in the server, the server will respond with an “Unsupported event” message and will not accept the Publish message. ‘Content-Type’ field indicates the format of the body which carries context information, and as described before we are using PIDF, so the content-type is “application/pidf+xml”. The header of the Publish message ends with the ‘Content-Length’ field which holds the length of the body “504” bytes. The remainder is the body of message in PIDF format. This body is used to send context information (see section 5.2.2 and 3.4.3.1).

To indicate that the Publish request was successfully received, the server will respond to this Publish message with a 2xx OK message as follows:

The Ok message in the reply of the Publish message, M2:

SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.196:5060*;branch=z9hG4bKqeAQbxW;received=*130.237.15.196
To: < sip:ccsleft@130.237.15.238>*;tag=a6a1c5f60faecf035a1ae5b6e96e979a-9247*
From: <sip:ccsleft@130.237.15.238>*;tag=qeAQ*
Call-ID: 78@192.168.1.196
CSeq: 1 PUBLISH
Expires: 2
SIP-ETag: 0xb58d412cx5184925bx47148a90
*Contact: <sip:*130.237.15.238:5060>
Server: Sip EXpress router (0.10.99-dev35-pa-4.2 (i386/linux))
Content-Length: 0

2.4.1.2 Subscribe & Notify

As described in section 2.4 Watchers use the Subscribe message to subscribe to events, hence they can be notified of changes in that event, as published by a PUA [27]. For example in Yu Sun‘s application [64] Watchers (or more precisely an application on behalf of Watchers) subscribe to the server for a location event in order to request the user’s location. Whenever the server receives a well formed Subscription, it will reply with an OK message (to indicate it received and can handle the subscription message successfully). Additionally, the server sends an immediate Notify message (to indicate the current state of the requested event). When the Subscribe expires, the server will remove that Watcher from its database and will inform the Watcher with a Notify message. Therefore, if the Watcher is still interested in this event, it has to send a new Subscribe message to the server.

The Subscribe message has been sent by the Watcher to the server, M3:

SUBSCRIBE sip:ccsleft@130.237.15.238 SIP/2.0
Via: SIP/2.0/UDP 130.237.15.238:5060;branch=z9hG4lfOiPzxGo
To: < sip:ccsleft@130.237.15.238>
From: <sip:Presence1@130.237.15.238>;tag=naVc
Call-ID: 238@130.237.238.165
CSeq: 1275 SUBSCRIBE
Max-Forwards: 70
Event: location
Accept: application/pidf+xml
Contact: <sip:Presence1@130.237.238.165>
Expires: 800
Content-Length: 0

In The first line, ‘SUBSCRIBE’ illustrates that it is a Subscribe message. As with the Publish message there are two variables in this first line, the host name “ccsleft” and the IP address (or Domain name) of the server “130.237.15.238”. The second line shows that “UDP” is being using as transport protocol. There are three variables in the second line, the IP address and port number of the server, where the subscription is to be sent (130.237.15.238:5060), and the ‘branch’ (as with the Publish message this is a random

number which should be different in different Subscribe messages). The "To" field contains the host name and the IP address of the server, whereas the "From" field indicates the host name of the Pocket PC (Presence1) with the IP address of the server. This means that Presence1 must register with the SIP registrar at the indicated IP address. Hence this device can be contacted via its SIP proxy. Again like 'branch', the 'tag' is a random number. In the next line 'Call-ID' is a field that has two parts, the first part needs to be a random value (in this case "238") and the second part is the IP address of the watcher. This field is used to distinguish separate requests within a conversation. 'CSeq' and 'Max-Forwards' fields are similar to the ones in Publish messages. In the next line 'Event' field indicates that Watcher interested in receiving the Notify (context information) message for the specific event, whereas here it is "location". The server uses this 'Event' field in order to decide when to notify this Watcher, hence a notify will only be sent when it receives a Publish message with the same event. If a Watcher asks for an event which is not defined in the server, the server will respond with "Unsupported event" message and will reject the Subscribe message. The 'Accept' field indicates that the Watcher is interested to receive the requested context information (via the body of Notify message) in a specific format, which it is PIDF in our case; therefore the content-type is "application/pidf+xml". The 'Expires' value should set by the Watcher to indicate how long this Subscribe is valid (in the ser.cfg file, the server can limit the maximum Expires values, see the Appendix E). Here in this message the expire time is "600", which translates to 600 seconds (see section 6). The header of each Subscribe message ends with the 'Content-Length' field which is always set to "0" because the Subscribe message does not have a body.

To indicate that the Subscribe message was successfully received, the server will respond to this Subscribe message with a 2xx OK message. An example of a response is shown in message M4 below.

The Ok message in the reply of the Subscribe message, M4:

SIP/2.0 200 OK
Via: SIP/2.0/UDP 130.237.15.238:5060;branch=z9hG4lfOiPzxGo;received=130.237.238.165
To: < sip:ccsleft@130.237.15.238>;tag=a6a1c5f60faecf035a1ae5b6e96e979a-0a75
From: <sip:Presence1@130.237.15.238>;tag=naVc
Call-ID: 238@130.237.15.238
CSeq: 1275 SUBSCRIBE
Expires: 600
Contact: <sip: 130.237.15.238:5060>
Server: Sip EXpress router (0.10.99-dev35-pa-4.2 (i386/linux))
Content-Length: 0

After sending the OK message, the server will send a Notify message to indicate the current state of this event. Notice that if there is no information about that event in the server at this time (i.e. because the previous Publish message has expired or the server has not yet received any Publish messages concerning this event), then the server will send a Notify with the "closed" value for <basic> tag in the body and no context information in the body. Such as Notify message is shown below.

The immediate Notify message when there is <u>no</u> context information on the server, M5:

```
NOTIFY sip:Presence1@130.237.238.165 SIP/2.0
Via: SIP/2.0/UDP 130.237.15.238;branch=z9hG4bKd856.f383a4c7.0
To: < sip:Presence1@130.237.15.238>;tag=naVc
From: <sip:ccsleft@130.237.15.238>;tag=a6a1c5f60faecf035a1ae5b6e96e979a-0a75
CSeq: 1 NOTIFY
Call-ID: 238@130.237.15.238
Content-Length: 211
User-Agent: Sip EXpress router(0.10.99-dev35-pa-4.1 (i386/linux))
Event: location
Content-Type: application/pidf+xml;charset="UTF-8"
Contact: <sip: 130.237.15.238:5060>
Subscription-State: active;expires=600
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf" entity=" pres:ccsleft@130.237.15.238">
<tuple id="none">
<status><basic>closed</basic></status>
</tuple>
</presence>
```

If there is some information about that event in the server, then the server will send a Notify with the related context information in the body. An example of such a Notify is shown below.

The immediate Notify message when there is context information on the server, M6:

```
NOTIFY sip: sip:Presence1@130.237.238.165 SIP/2.0
Via: SIP/2.0/UDP 130.237.15.238 ;branch=z9hG4bK125c.b5adefb3.0
To: <sip: Presence1@130.237.15.238>;tag=Dved
From: <sip:ccsleft@130.237.15.238>;tag=a6a1c5f60faecf035a1ae5b6e96e979a-3a6b
CSeq: 2 NOTIFY
Call-ID: 248@130.237.15.238
Content-Length: 546
User-Agent: Sip EXpress router(0.10.99-dev35-pa-4.2 (i386/linux))
Event: location
Content-Type: application/pidf+xml;charset="UTF-8"
Contact: <sip:130.237.15.238:5060>
Subscription-State: active;expires=373
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf" entity=" pres:ccsleft@130.237.15.238">
<tuple id="0xb58d60e0x4a4b0c39x4715c26e">
<status><basic>open</basic>
<location>
<description>Electrum</description>
<room>Wireless</room>
<floor>1</floor>
<coordinates>
<latitude>47382145</latitude>
<longitude>123213</longitude>
<height></height>
</coordinates>
</location>
</status>
<contact priority="0.80">Mohammad</contact>
<note>location</note>
</tuple>
</presence>
```

In both cases, in order for the Watcher to indicate that has received the Notify message, it will send a 2xx OK to the server as follows:

The Ok message in the reply of the Notify message, M7:

SIP/2.0 200 OK
Via: SIP/2.0/UDP
130.237.15.238;branch=z9hG4bKd856.f383a4c7.0;received=130.237.238.165
To: <sip:Presence1@130.237.15.238 >;tag=naVc
From: <sip:ccsleft@130.237.15.238>;tag=a6a1c5f60faecf035a1ae5b6e96e979a-0a75
CSeq: 1 NOTIFY
Event: location
Content-Length: 0

2.5 CPL

The Call Processing Language (CPL) is a language that can be used to describe and control Internet telephony services. It is described in RFC 3880 [40] and was designed to be implemented either on network servers or user agent servers, as both can usefully process and direct an incoming call to a user.

CPL is an XML-based language. It is simple, extensible, easily edited by graphical clients, and independent of the operating system or underlying signaling protocol. It is suitable for running on a server, because it is not Turing complete. This means users, may not execute arbitrary programs or anything complex, as this programming language has no variables, loops, and lack the ability to run external programs. Additionally it does not support recursion.

2.5.1 Network model

The internet telephony network model for CPL consist of two main components: End systems and signaling servers [41]. Each of these will be explained below.

End systems: A device from which and to which calls are established. It originates or receives signaling information and media (audio, video, or the like). An end system can originate, accept, reject a call, or forward incoming calls. Examples of such end systems include: telephone devices, PC telephony clients and automated voice systems.

Signaling server: A device which handles, relays, or controls signaling information. It does **not** process or interact with the media of a call. In SIP, a signaling server may be a SIP proxy, redirect server, or registrar. Usually, a signaling server can perform some actions on the call signaling: it can forward it to one or more other servers or end systems, returning one of the

responses received (i.e. acting as a proxy), return a response informing the sending system of a different address to which it should send the request (redirect it), or it can inform the sending system that the setup request could not be completed (reject it). Additionally a signaling server can store user location information: obtained via SIP client registration, thus determining the user's current SIP agent location. The signaling server can generate a transaction log: by the means of storing the information which passes through it, or send email to notify users of change in state of the signaling server or a call.

When an end system places a call, the originating end system must decide where to send its requests. The originator may send all requests to a single local server; or it may resolve the destination address in order to send the request directly. Once the request arrives at a signaling server, this server uses its user location database, local policy, DNS resolution, or other methods, to determine the next signaling server or end system to which the request should be sent. A request may pass through any number of signaling servers: ranging from zero (in the case when end systems communicate directly) to every server on the network.

2.5.2 CPL: basics

CPL was designed as executable language to allow an untrusted user to upload services to be run on a SIP proxy server (e.g. SER). CPL, like SIP, is a text-based protocol. CPL is an official work item of the IPTEL WG, and it is based upon the XML Language. XML tags have the form of <tag>, which opens the tag, followed by </tag>, which closes the tag [13]. The Document Type Definition (DTD) for CPL is specified in the "cpl.dtd" file, available at [42]. "cpl.dtd" specifies the syntax. CPL scripts should be validated (syntax check) like any other XML DTD files, before they are uploaded to the server. Some tags have attributes, in which case they are written as <tag attribute="value">. Tags also can have multiple attributes. Tags without any attributes, or nested tags, can be opened and closed in a single tag using <tag />, which is equivalent to <tag></tag>.

Despite the fact that, CPL was designed for end users to create services, other parties can create CPL as well. For example, a third party or an administrator can use CPL to customize services for clients. In addition, CPL scripts can be created on end user devices. This device need not be the device to or from which a call is placed/received. For instance, scripts can be created on a PC, while the call could occur using a physical phone. This is supported because after creating a CPL script, it should be uploaded to the SIP server. Although there is no specific method for CPL script uploading, this upload can be realized in a secure manner, depending on the server. Not only is the confidentiality of the request required, but also it must be authenticated, thus only the legitimate user can redirect or otherwise process their own call(s). After being uploaded, the script can be run on the servers owned by end users or service providers. Moreover, there are many possible ways to create CPL scripts. Users can create CPL by writing code, but this method is not user friendly, so the user may want to create CPL using GUI tools, such as CPLEd. The CPLEd is a java application with a graphical interface and can be used to create, edit, or upload CPL scripts to the SIP server.

As shown in Figure 4, each SIP server has a database of CPL scripts, when a session (call) establishment request arrives, the server utilizes the source and the destination address to look up the applicable CPL script in a database, if a match occurs, and then the corresponding script will be executed. If there is no match, the signaling server will utilize the default location lookup service. When Alice wants to make a phone call to Bob, she sends the SIP INVITE message to a SIP proxy (initiating an in/outgoing call), then the server executes the appropriate part of the user's CPL script that matches with this call, then it makes a decision (accept and route the call to callee, reject the call, forward it to voicemail, redirect, etc.) [43]. Furthermore, CPL enables the SIP server to provide many features, such as call blocking, call redirecting, etc., based upon the context information such as time, caller, callee, etc. These functions can be applied to both outgoing or to incoming calls.

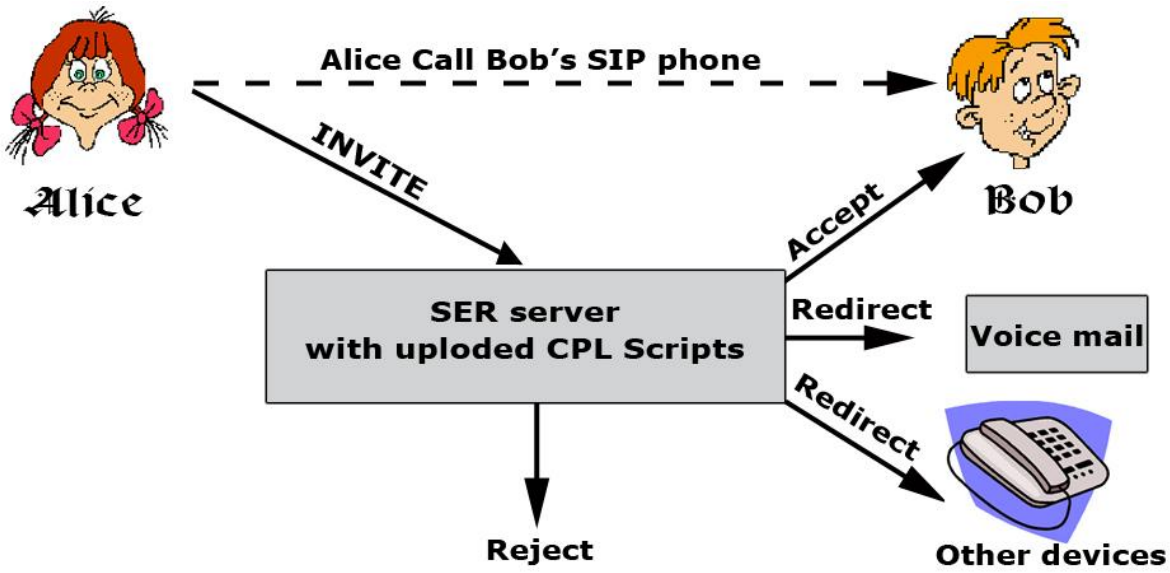

Figure 5. SER decisions based on CPL scripts

2.5.3 Components

A CPL script consists of two parts: Ancillary information and Actions. Ancillary information does not directly describe operations or decisions. However, but this information is necessary for a server to correctly process a script. An **action** is a structured tree that describes the operations and decisions which a signaling server will perform on a call set-up event (an incoming or outgoing call). Actions are further divided into sub-actions and top-level actions [41]:

- Top-level actions are actions that are triggered by signaling events that arrive at the server. Two top-level actions are defined: "incoming", the action performed when a call arrives whose destination is the owner of the script, and "outgoing", the action performed when a call arrives whose originator is the owner of the script.

- Sub-actions are actions, can be initiated by other actions and may **not** be called recursively.

As shown in Figure 5, a CPL script is a collection of: nodes which describe operations, decisions, and outputs. Nodes may have one or more parameters (to specify the behavior of the node) and one or more outputs (depending on the result of a decision or action). Nodes and outputs are both described by XML tags. Outputs of nodes are connected to other nodes. When the action of the top-level node is invoked, based on the result of that node a server utilizes one of the node's outputs, and the subsequent node it points to is invoked. This procedure is repeated until a node with no outputs is reached [43].

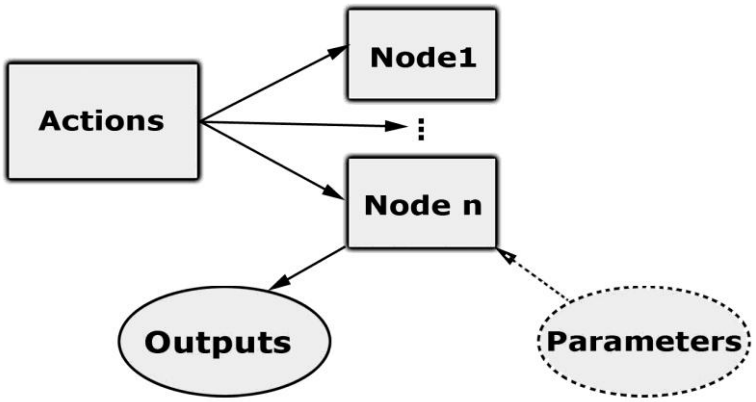

Figure 6. CPL components

There are four categories of nodes [41]:

Switches symbolize decisions a CPL script can make (based on either attributes of the original call request or items independent of the call).

Location modifiers add or remove locations from the location set.

Signaling actions cause signaling events in the underlying protocol.

Non-signaling actions take an action that does not affect the underlying protocol.

Switches are further divided into:

Address allows a CPL script to make decisions based on one of the addresses present in the original call request.

String allows string matching on a string variable, in order to make decisions based on this information such as: language, display, user-agent, organization, and subject (e.g. this allows the client to make decisions based on the language which this caller wants to receive)

Time allow a CPL script to make decisions based on the time and/or date, for example only accepting specific calls Monday to Friday 8:00-17:00.

Priority allows a CPL script to make decisions based on the priority specified for the original call.

Each of these switches has a number of attributes, including fields and subfields. The matching rules include “is”, “contains”, and “subdomain-of”. The complete set of switches is specified in [13].

Because CPL actions may by dependent upon the current location, Location modifiers add or remove locations from the location set. These modifiers are divided in to two categories [41]:

- **Explicit location nodes**: specify a location internally.
- **Location lookup:** specify a location through external means.

2.5.4 Two Examples of CPL scripts

In this section we show some example CPL scripts, including a description of the tags which are utilized in each script.

2.5.4.1 Redirecting an incoming call

In this example any incoming call will be redirected to the location specified by the URI: sip:mohammad@kth.se. The XML code to do this is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cpl PUBLIC '-//IETF//DTD RFCxxxx CPL 1.0//EN' 'cpl.dtd'>
<cpl>
<incoming>
<location url="sip:mohammad@kth.se">
<redirect permanent="yes" />
</location>
</incoming>
</cpl>
```

The first tag indicates the version of XML (in this case version 1.0), whereas the second tag starts the CPL script, indicating the XML namespace, and defines the schema, which supplies the parsing rules for the document. Everything between <cpl> and </cpl> tags is the CPL script. The next tag <incoming> indicates that this script has been written for incoming calls, rather than an outgoing one. The next tag is <location>, which determines to which URI (where) the incoming call should be redirected. The <redirect> tag with the attribute, permanent, indicates that the incoming call should always be redirected to the location specified. The rest of the tags simply close the open tags.

2.5.4.2 Rejecting outgoing calls destinations start with 0046

In this example any outgoing call to the destination which starts with 0046 will be rejected:

```
<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
<outgoing>
<address-switch field="original-destination" subfield="tel">
<address subdomain-of="0046">
<reject status="reject"
reason="Not allowed to make calls to Sweden."/>
</address>
</address-switch>
</outgoing>
</cpl>
```

The first and second tags are same as previous example, while in contrast the <outgoing> tag indicates that this script defines behavior for outgoing calls. The next tag is <address-switch>, which is a type of switch or decision point, which indicates the username part of the origin destination address (From header) is the value being tested. The <address> tag with the attribute subdomain-of=”0046” means that any telephony URIs to Sweden will be rejected. While the <reject> tag with its attributes, status, and, reason, indicates that when an outgoing calls that matches the condition it should be rejected. The rest of the tags simply close the open tags.

You can find additional sample CPL scripts in [13] and appendix D in [21].

2.5.5 CPL extensions

There are services, such as context aware services relevant to VoIP, that are difficult or impossible to implement with basic CPL. CPL extensions have been proposed to provide the ability to implement these services. As shown in Table 1, Alisa Devlic [43] added and defined context parameters such as: context owner, his/her location, task, and activity in a context-switch CPL switch in order to implement context aware services using CPL. The definition of CPL these extensions should be specified in the "context.dtd" [44]. Additionally the cpl-c module of the SER source code had to be modified to support the addition of this context-switch and context node.

Table 2. CPL extension for context aware service [43]

Node name	Node type	Parameters	Description
Context-switch	Switch	Owner	Owner of context
Context	Output of Context-switch	Location	Location of Context owner
		Task	Task Status
		Activity	Activity Status

As an example of extended CPL, when Mohammad (an end user) has an incoming call while having his lunch at the Electrum building restaurant, his SIP server will utilize the script, which he has previously uploaded, to redirect this call to his voice mail:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cpl SYSTEM
'file:C:/Programs/CPLEd/context.dtd'>
<cpl>
<incoming>
<location url="sip:mohammad@example.com">
<context-switch owner="Mohammad">
<context location="Office" task="In the Electrum restaurant"
activity="Eating lunch">
<redirect status="redirect" reason="I am in the Electrum restaurant having lunch"/>
</context>
</context-switch>
</location>
</incoming>
</cpl>
```

3 Context Aware Services

3.1 Context definition

Schilit and Theimer defined context as location in terms of "Context aware" applications, this in turn identities nearby people and objects and changes to those objects [32]. Brown, Bovey, and Chen also define context as location, but focus on the people around the user, the time of day, the season of the year, the current temperature, etc. [33]. However, today we must consider new types of context information. However, these definitions do not cover all types of context; for example, currently we do not consider user preferences as an element of user context.

Schilit, Adams, and Want introduce three important aspect of context [34]: where the user is, who the user is with, and what resources are nearby. Overall it seems Dey's definition is more meaningful and close to my own idea: "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" [35]. With this definition, almost all information that occurs in the context of a specific system usage can be subsumed under the term "context".

After defining context, context **usage** can be divided in the number of categories as follows [2]:

- Presenting the context information as content to the user: for example presenting a choice of printers close to the user.
- Automatically executing a service, triggering actions, or reconfiguring a system on the occurrence or change of a context: for example sending reminders when a user is in the specific location (see Yu Sun master thesis [64])
- Deciding and performing an action on behalf of the user based on the context information and user preferences: for example rejecting a phone call when a user is on the meeting.
- Attaching context to information for later retrieval: for example attaching date/time/weather information when taking a picture for advanced process of printing picture later.

3.2 A Context aware scenario (context-aware printing system)

In this section a simple scenario will be described, in order to illustrate how a context aware infrastructure serves a real context aware application. This sample application is a context-aware printing system, whereas the application finds available printers on a university

campus and recommends the appropriate printer to the user. This application will utilize context information such as the user's location, user preferences, document type, printer's tuner/ink/… status, etc. in order to recommend one or more printers to the user. Assume that Alice wants to utilize this application, and a printer "printer1.wireless.kth.se" changed from "ready" to "out of paper":

1. In order for Alice to subscribe to this service, Alice's local application will authenticate and register, thus creating an entry in the SIP server's database, which is actually a local MySQL database that SER uses. She will subscribe via a context-aware printing service module, in order to be notified of printer related context information.

2. A service agent, which is implemented by software installed on a PC connected (via Ethernet or parallel port) to this printer, will register with SER to update the server's knowledge about changes in this printer's context information via the SIP-SIMPLE method (which is implemented as a new SIP-SIMPLE software module for SER).

3. This service agent periodically acquires relevant printer related information through PJL or MIB queries (details of these are given in Appendix C). This information includes: printer status, printer errors, ...

4. Now assume the printer's status changes from **ready** to **out of paper**, then the service agent will inform SER of this change in the printer's state through SIP-SIMPLE. Subsequently when Alice asks about this printer's status through a SIP request message, SER will reply with a SIP response message indicating that this printer is currently out of paper. Note that asynchronously notifying Alice about the printer's status changes can be done also by SIP-SIMPLE method instead of a SIP request/reply.

In all four steps mentioned above, in order to store, publish, exchange, and represent printer context information properly, either an extension of PIDF or PWG models can be used as a context model (Note that these two models are discussed in sections 3.4.3.1 and 3.4.3.2).

3.3 Architecture

Different types of related technologies were briefly described in the previous sections. Context-aware systems can be implemented using a number of different architectures. Several criteria should be considered in order to choose an appropriate architecture, such as [1]: the geographical location of elements (both users and sensors), the number of users (which will affect the traffic of system), the available resources of the devices to be used (less resources on user device side translates to more work on system side), the number of supported

applications, and desired future extensibility of system. Moreover every context system, first needs to acquire context information typically via sensors (hardware or software sensors), then store and process this context information, and finally distribute it to different applications and users who is interested in this context data. These four processes (context acquisition, storing, processing, and distributing) should be considered when designing an architecture. Chen [5] describes three different types of context architectures: Direct sensor access, Middleware infrastructure, and Context Server. Each of these will be described below.

Direct sensor access: As we can see in the Figure 6, in this approach there is no specific layer for acquisition of context information and each application gathers the desired context information directly from sensors. Although a simple architecture, an implementation of such a system is easy, it has some deficiencies such as: it is unsuitable for distributed systems (due to applications needing to directly access sensors), it is difficult to add new sensors or applications (poor extensibility), there is limited reuse of context information, and it supports only a limited number of users, sensors, and applications (poor scalability). These limitations generally cause designers to avoid using this architecture.

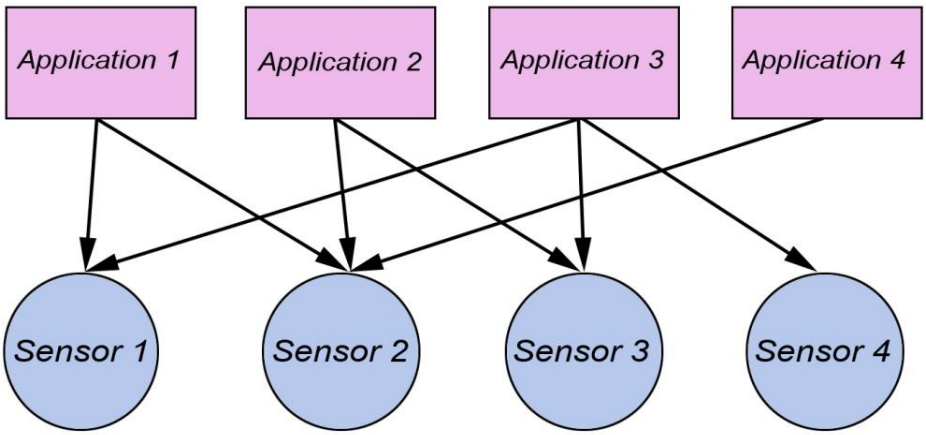

Figure 7. Direct sensor access architecture

Middleware infrastructure: this approach introduces a layered architecture, which is used by the majority of modern context aware systems. Although designing and implementing this architecture is more complicated than the direct sensor access architecture, it supports better scalability, extensibility, and reusability. As shown in Figure 7 the first layer (from the bottom) collects context information from the environment (sensors) and delivers it to higher layers which are responsible for storing context information in a context repository. This repository stores a collection of arbitrary context objects. Higher layers process this context information and finally the highest layer distributes the context information to the different applications, service agents, or users desiring it. This distribution can occur in one of two ways: (a) a subscription-based push method (in this thesis we are using SIP-SIMPLE, see section 2.4) which provides asynchronous access to context or (b) in a synchronous pull-based manner (SIP request-response, see section 2.4) which directly queries for context by sending request- response messages.

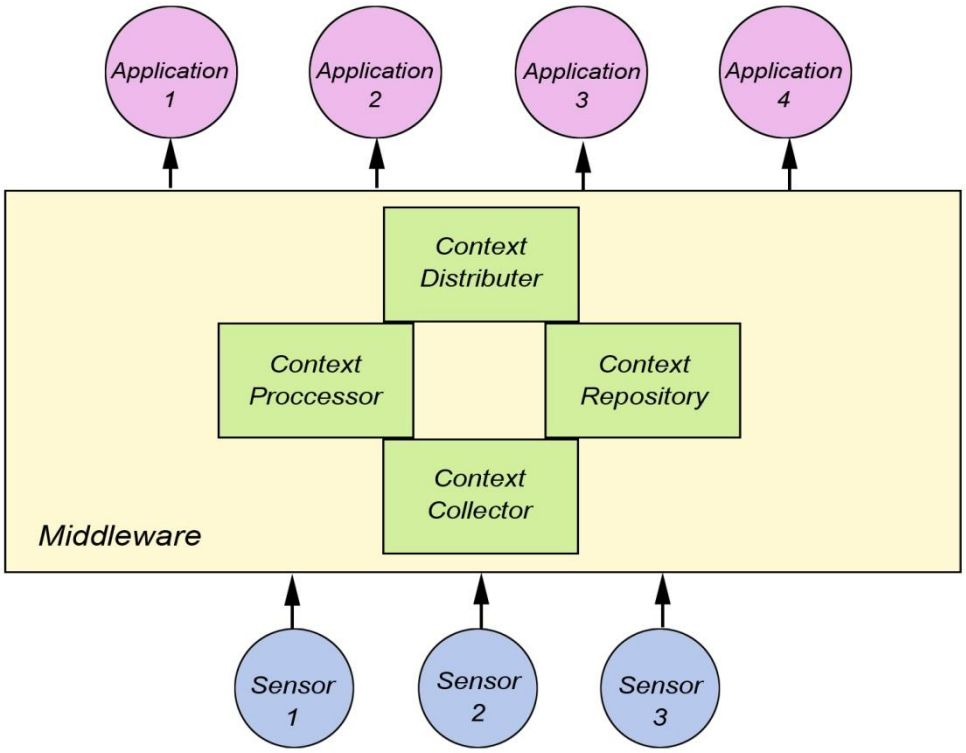

Figure 8. Middleware architecture

Context Server: In this approach a context server is added to the Middleware architecture to allow multiple applications to reuse sensor data. This is an important factor as most of the client devices have limited resources. Although this architecture offers higher performance, implementation of it is more complicated. Such an architecture has many parameters, such as: appropriate protocols, processing power, network performance, quality of service parameters, etc. Each of these needs to be considered to create a suitable design and implementation.

Comparing theses three proposed architecture shows that: (a) the direct sensor access suffers from poor extensibility and scalability, (b) although context server architecture eases the applications from the intensive computations, it's desidn is complicated compareing to middleware architecture. Therefore for this master thesis, the middleware architecture has been choosen.

3.3.1 Related work

A number of Infrastructures have been implemented based on the two last architectures described in the previous section. Some of these are described below:

Widgets: The context widget was introduced by Dey, et al. [3] in their software toolkit. As shown in the Figure 8, this toolkit provides an interface for a hardware sensor, which is

responsible for obtaining context information from sensors, and making it available to applications via polling or pushing based methods. This toolkit facilitates the development task for application developers and hides the complexity of low level details of getting information from sensors (applications can request this information, if desired). Moreover a widget abstracts context information to interested applications, thus sending only significant context information to them, and also provides reusability of context information for different kinds of applications. This software component includes four additional categories: Interpreters (accept one or more context sources, and produces a new single more abstract piece of context information), Aggregators (collect related context about an entity from widgets and interpreters, on behalf of the interested applications), Services (execute actions on behalf of applications in order to control or change state information in the environment), and Discoverers (maintaining a registry of what capabilities exist in the framework, including what widgets, interpreters, aggregators and services are currently available, and enable applications to locate context components that are of interest to them).

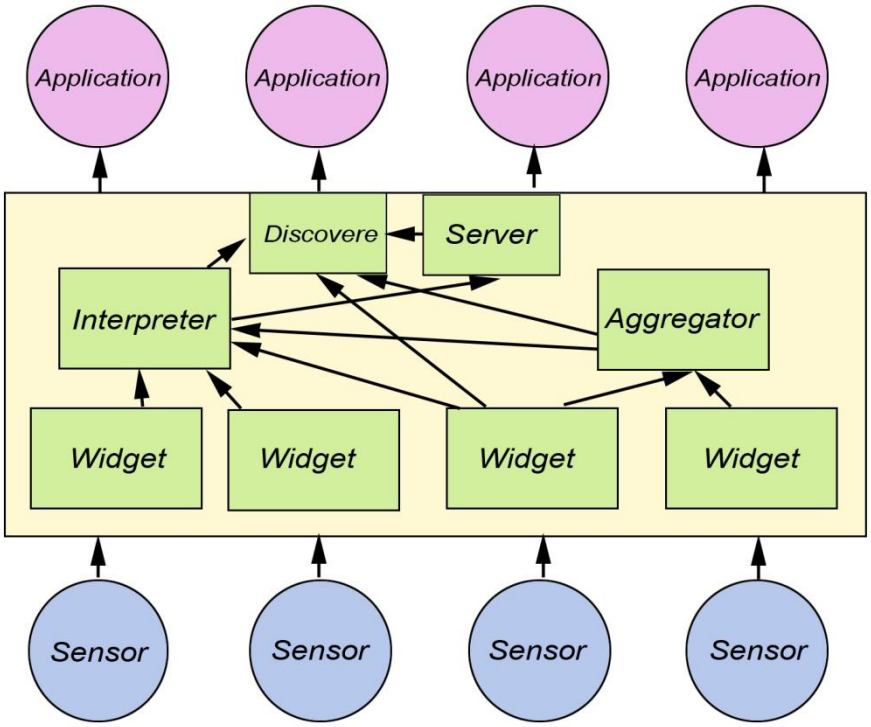

Figure 9. Widget toolkit

Network middleware: This approach is similar to the Context Middleware architecture, but unlike the widget method (which used a process-centric model) this approach uses a service-oriented model. Two infrastructures based on this method are described in Capra et al. [25] and Hong & Landay [26]. Network middleware works on top of a network operating system and provides application developers with higher levels of abstraction, hides complexities, and discovery techniques are used to find networked services, rather than using a global widget manager.

Context Broker Architecture (CoBrA): This Infrastructure, by Chen et al. [9] is based on a Context server architecture. This asymmetric model represents a data-centric model, where a unit of data is gathered from all types of sensors (e.g. temperature sensor, presence detector, and etc.). This architecture provides a shared model of the context data for all services, applications, and users; therefore it supports more distributed reasoning capabilities. Although this model simplifies adding new context sources, a centralized server is required to host the Broker agent.

ACAS middleware: The Adaptive & Context-Aware Services (ACAS) project by Jansson et al. [59] at Wireless@KTH investigated the use of context-awareness in order to provide efficiently delivery of services to users who move about in a wireless heterogeneous infrastructure (e.g. IEEE 802.11 WLAN access, 3G, etc.). They extended a SIP-SIMPLE service network for their infrastructure in order to publish and provide context information to interested entities. Context Management enabled Entities (CMEs) were proposed as the basis of a context infrastructure, which connects to various applications; then in order to provide relevant context information to both applications and devices, these context management enabled entities can be interconnected to each other. Moreover, CMEs contain a context server (make decisions of whether to serve a request or not, based on service policies), and a context manager (processes and manages context information, in order to provide context information for requests that have been passed through the context sever). CMEs use SIP for addressing, and Context Data eXchange Protocol (CDXP) [60] for carrying context information directly between hosts.

3.4 Context Modeling

3.4.1 Why Modeling

Context modeling is needed to describe, represent, exchange, and store context information in a way that it can be understood and processed by different types of computer (e.g., PCs, PDAs, etc.). Selecting and extending an appropriate model which covers all possible types of context information is an important issue in developing a context aware application. The following sections describe some of the basic context models, as summarized by Strang and Linnhoff-Popien[18]:

3.4.2 Different methods for modeling

3.4.2.1 Key-Value models

Key-Value models are simplest and most widely used form of modeling (but unfortunately inefficient for context retrieval), in different context aware applications. Key-value pairs are used to describe the context information, where the key represents the type of context and the value contains the value of this context. For example: context Status =

(StatusPrinter1, Busy). In this example “Status” is the key, which means this context holds the printer’s status information; “StatusPrinter1” and “Busy” are context values indicating that priner1 is busy now.

3.4.2.2 Graphical Models

Extensions of the Unified Modeling Language (UML) [20] and the Object-Role Modeling (ORM) [37] are the examples of graphical models. Examples of this modeling approach can be found in [11] (UML) and [36] (ORM). Although graphical models are easily readable by humans, they require more work to implement and evaluate [1].

3.4.2.3 Object oriented models

Object oriented models use the efficiency of object orientation designs (e.g., encapsulation, reusability, and inheritance). Various objects can be used to represent different context types (i.e. printer status, printer location, etc.). An example of using this model can be seen in the Hydrogen project [38].

3.4.2.4 Logic models

Logic models for context define context as facts, experiments, and rules; and a logic based system can be used to add, update, or remove facts. Logic based models have a high degree of formality, but these models suffer from difficulties in determining validity [1].

3.4.2.5 Markup scheme models

A hierarchical data structure consisting of markup tags with attributes and elements are used in a markup scheme model. Profiles are typical markup-scheme models (i.e. the Composite Capabilities/Preference Profile (CC/PP) [39] designed by W3C, User Agent Profile (UAProf) [45] defined and maintained by the Open Mobile Alliance. Both CC/PP and UAProf are vocabulary extension of the Resource Description Framework (RDF) - , and etc.).

Due to several attractive features of XML language (see section 2.2), generally an extension of XML is used for markup scheme models. Although XML DTDs and XML Schemas are sufficient for exchanging data in a domain (where elements have been defined beforehand), their lack of semantics (e.g., meaning) prevents machines from easily describing context data. RDF [46] (which uses the XML data format to describe resources) and RDF Schema (which defines the vocabulary used in RDF data models) were designed by W3C to solve this problem by allowing simple semantics to be associated with identifiers. Using an RDF Schema, one can define classes, properties, domains, and ranges; therefore an RDF Schema can be considered a simple ontology language (see section 3.3.7). An RDF document consists of statements, each consisting of a triple: a subject (resource), a predicate (property), and an object (a value assigned to that property). The example below illustrates how the status of a printer could be described in RDF:

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sensor="http://www.it.kth.se/~moze/printerd#">
<rdf:Description
 rdf:about="statusPrinterCCSLab1">
   <sensor:statusReading>online</sensor:statusReading>
</rdf:Description>
</rdf:RDF>
```

3.4.2.6 Ontology based models

In context aware systems, an ontology defines a common vocabulary (to enable understanding of a context information) for users, applications, and infrastructures (databases), which wish to share information in a domain. An ontology also enables reuse of the context information in the same domain. Assume that different printers use different vocabulary for being “out of paper”, but in this thesis we define “OutOfPaper” as a word in the vocabulary in the “KTH” domain which means (in this domain) that a printer is out of paper.

Moreover, an ontology includes machine understandable definitions of basic concepts in the domain and relations among these concepts. Once an ontology is developed, others can simply reuse it if it is applicable for their domain. Additionally, several existing ontologies can be integrated, or a general ontology can be extended, in order to describe context information in a specific domain. Therefore, ontologies are well suited for context-aware systems and various context-aware frameworks use ontologies as their underlying context models. The Web ontology language (OWL) [47] is an example of one of the ontology languages designed by W3C. It is built on top of RDF and RDFS to add more vocabulary (OWL was derived from the DAML+OIL Web Ontology Language).OWL provides great flexibility for describing resources. OWL is designed to be used when the information contained in documents needs to be processed by applications (not only when it needs to be presented to humans). OWL can be used to represent the meaning of terms in vocabularies and the relationships between those terms in a machine-understandable way.

The semantic web [48] is an extension of the present World Wide Web, designed by W3C in order to enable computers to be able to reason about web information in addition to displaying web information. The semantic web uses RDF/RDFS (to represent web information as a resource) and OWL (to define ontology vocabularies).

3.4.3 Candidate models

After briefly describing different several approaches for context modeling, a method must be selected to use in our infrastructure. Although each context aware system has its own properties and needs for context modeling, it seems that the markup scheme and ontology based models, when compared to the other methods, are more matched to this thesis context

modeling requirements, and mostly used in different applications. Given the context aware service requirements of this thesis, we chose to use two different methods for context modeling based on a markup scheme method: Presence Information Data Format (PIDF) (due to fact that SIP-SIMPLE has been chosen for distributing context information among entities, PIDF can be used as one of the context models) or Printer Working Group (PWG) semantic model (this model is suitable for context-aware printing application). Each of these methods is described briefly in the following sections:

3.4.3.1 PIDF

As described earlier in section 2.4, SIP-SIMPLE supports presence functions, by providing a method for users to subscribe to events, in order to be notified of changes. We have decided to use SIP-SIMPLE to distribute context information among entities. Hence a context model is needed to transfer this context information (presence documents) in a standard format. The Presence Information Data Format (PIDF), as defined in [29], provides a means for transferring presence information in a domain without modification and with high performance. PIDF has been designed to be extensible and flexible, thus a presence application is able to define its own status values.

A presence document encodes presence information as a well formed XML document, thus it must have an XML declaration (e.g. "<?xml version='1.0' encoding='UTF-8'?>") [1]. PIDF covers and extends the minimal model of Instant Messaging and Presence Protocol (IMPP) [51]. PIDF has some basic elements used in presence documents within the XML namespace name 'urn:ietf:params:xml:ns:pidf'´. These basic elements are described briefly below.

Presence Is a root element and contains all other elements. The <presence> element **must** have an 'entity' attribute, containing the URI of the presentity that published the presence document. Moreover, the <presence> element **must** contain a namespace declaration ('xmlns') to indicate which presence document it is based on.

Tuple Each presence document contains any number[1] of <tuple> elements, which describe status information. These elements consisting of a mandatory <status> element potentially followed by any number of optional known elements (e.g. contact, note, and timestamp) or extended optional elements (from other name spaces). Moreover, each <tuple> element has a unique "id" (which differentiates it from the other <tuple> elements in the same presentity). This id is used to segment the presence information into different tuples (instead of creating multiple PIDF instances). This is done in order to differentiate presence information (e.g. associated with a given presentity).

[1] The number of tuples could be zero, for example, when the NOTIFY sent after an initial SUBSCRIBE request does not have any changes to report

Status Contains value elements (e.g. change in a printer status element) in order to describe the presence information. It is mandatory for each status element to have at least one value element. The <status> element may contain one OPTIONAL <basic> element, which indicates the availability of the remote entity to receive instant messages, and possibly a number of OPTIONAL extension elements (to fulfill different Instant Messaging application requirements). The <basic> element contains one of the following strings: "open" (i.e. ready to accept an instant message) or "closed" (i.e. unable to accept an instant message).

Contact Is an optional element, contains the URL of the contact address of the presentity (e.g. SIP URI, IP address, or etc.). It has an optional attribute 'priority', indicating the relative priority of this contact address over others.

Note Is an optional element, containing a string value, which is usually used for human-readable comments about presence state information or presentity. The <note> element has a attribute 'xml:lang', indicating the language used in the contents of this element.

Timestamp Is an optional element, containing a string value, and indicating the time and date of the status change of this specific tuple. Successive <presence> elements should not be created with the same timestamp by presentity.

Beside the defined elements described above, In order to be able to support a broad range of context information, other custom-built elements can be defined under the <status> element (extensibility). For example, to notify an application of changes in the status of printers in printing scenario, a <printerstatus> element can be defined:

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
    xmlns:printer="urn:ietf:params:xml:ns:pidf:printer"
    entity="pres:someone@example.com">
  <tuple id="ub13bu">
    <status>
      <printerstatus>out of paper</printerstatus>
    </status>
    <timestamp>2007-08-08T16:18:28Z</timestamp>
  </tuple>
</presence>
```

Note that beside fundamental name spaces, a *'xmlns:printer'* name space was used. The <printerstatus> element is defined in this name space beforehand. Moreover, a <note> tag can

also be used to indicate the status of the printer. A note tag would be used when the context information is only a simple text (less information in simple formats) and there is no reason for creating new tags for that purpose, therefore it is easy to only put the context information in the note tag rather than creating a new one. Besides extending different elements under the <status> element to support different application owners from other name spaces, developers may extend the basic format of PIDF in order to convey richer presence information. These extensions are: RPID, PIDF-diff, and Timed Presence-extensions to PIDF. Each of these is described below.

RPID The Rich Presence Information Data format [52], defines additional presence attributes to describe person, service, and device data elements. Beside the <tuple> fundamental element, additional elements are added, such as <device> and <person> elements, in the data model, in order to indicate logical properties such as user mood and activity, as well as physical properties such as location and environment. This approach is a fusion between presence information and data found in calendar systems, which are derived automatically data from other information sources, such as calendar files, the status of communication devices such as telephones, typing activity, and physical presence detectors.

PIDF-diff PIDF for Partial Presence [53] is an extension to PIDF. The problem with PIDF is that it carries full presence information in every single change or update in presence of an entity. As this can cause some problems (e.g. congestion) in environments with low bandwidth and high latency links, PIDF-diff was designed to address this problem by introducing a new MIME type which enables transporting either only the changed parts or the full PIDF based presence information.

Timed extension the Timed Presence extension to PIDF [54], adds a timed status extension (<timed-status> element), which allows a presentity to express past intervals, durations, and future intervals in relation to a presence property. Normally, PIDF shows presence information as a description of a current status; however, in some applications such as the printing scenario, it makes sense to express not only current status, but also to indicate the recent history of the status of a printer or perhaps to indicate the future status of this printer (e.g., when it will be available, or for how long it is busy). This additional information will be useful if we want to schedule the printing of a document in the (near) future.

3.4.3.2 PWG

In the last decade a group of companies including IBM, Microsoft, Novell, Epson, Canon, etc. recognized a need to create a standard for printing across the network, thus they initiated “The Printing Working group” (PWG) [55], creating what finally became the Internet Printing Protocol (IPP) [56]. This protocol allows users to print to a remote printer as

well as managing print jobs, media size, etc. The PWG Semantic Model (SM) [50] was defined by the PWG and is primarily based on the model used by IPP. PWG is a collection of XML documents extending the XML Schema, defining a simplified printing model, with a number of elements, in order to describe the printer, print jobs, and documents (a job can contain zero or more documents). An example, showing the status of a printer in this model is:

```
<PrinterStatus>
    <PrinterCurrentTime>1354</PrinterCurrentTime>
    <PrinterIsAcceptingJobs>true</PrinterIsAcceptingJobs>
    <PrinterState>Idle</PrinterState>
    <PrinterStateMessage>Ready</PrinterStateMessage>
    <PrinterUpTime>1023</PrinterUpTime>
</PrinterStatus>
```

3.5 Context Discovery for Printing Scenario (Getting Information from printers and setting printing preferences)

3.5.1 Which information is most relevant to our service?

In order to recommend the best printer to the user we need to collect some information about the document that the user wants to print (this information can be gathered from the user or a service agent which receives the user's document in order to print it). We also need to gather information about which printers exist; along with each printer's capabilities and configuration. Table 2 lists some of the information that we might want. How this information can be acquired and used is describe in the next sections.

Table 3. Information, needs to acquire from both the printer and user; in order to support context aware printing system

Information about the nature of the document and printing preferences (source: user)	**Information about the printer (source: service agent)**
Method: Software running on the device, PJL	Method: PJL, SNMP (MIBS)
Document title Document type (txt, jpg, pdf, etc.) Paper size Paper format Print in color/black-white Number of pages User's location Language Pending Jobs	Printer status Pages per minute (ppm) Toner status Color printing capability Document pages number Supported paper format Printer location Printer name & model Out of paper Page counter

3.5.2 Protocols and software tools

3.5.2.1 Network printers

Contemporary networked printers are usually accessible through a network. Although there are some tools supplied by vendors which simplify acquiring information from printers, such as HP's Web JetAdmin [57]; these tools are generally proprietary, binary applications, and there is no easy way to incorporate their functionality into other applications. In order to implement proposed the context aware application for printers, we wish to get information from printers via an application programming interface. We have chosen to use two different methods, namely Printer Job Language (PJL) [30] and SNMP MIBS (note: an existing tool using the later approach is npadmin [31]). These two methods enable us to get the information which we need (such as Printer status, memory, toner levels, etc.) for our printing context service during the context discovery phase. (The information concerning the printer is listed in the right most column of Table 2). To see how we get information from a printer by using these methods see appendix C.

4 Goals & Methods

We have chosen to use a middleware architecture, as explained in section 3.3, as our infrastructure. The design, implementation, and evaluation of this context middleware will be the main focus of this thesis project. To do this a SER server, MySQL database, and a context module will be used (the later is called a "presence agent"; it will be loaded along with other SER modules when SER starts). To simplify references to this, the whole system will be referred to as a **presence agent server** in this thesis. The overall architecture is shown in Figure 9. In this figure we can see that this server is responsible for:

1. Obtaining raw context information from Presence User Agent (PUAs) or service agents by using Publish messages (described in section 2.4.1.1). (Note that PUAs are responsible for obtaining data from sensors, however the details of this are outside scope of this thesis.) An example of a PUA which our server should deal with is "the occupancy sensor system for context –aware computing" (this is a room occupancy detector) being developed by Daniel Hubinette [63], and the context-aware printing application being developed by Athanasios et al. [65]. Whenever a change occurs in context, these PUAs will inform my server with the publish message.

2. Read and process this raw context data is required in order to: (a) store the valuable information for later usage in different applications and services, (b) eliminate valueless data, or (c) make decisions.

3. Store the information in the local MySQL database co-located with the SER server, for later usage.

4. Send the relevant context to interested users or services via Notify messages. This will be done via asynchronous subscriptions (i.e. using the SIP-SIMPLE), as this approach is most suitable for rapid changes in the context. Thus our server should work with the "Context-aware applications for a Pocket PC" application being developed by Yu Sun [64] which provides reminders based upon the user's location (not only the time of day). The server will inform the subscribed users (using a Subscription message) by sending a Notify message whenever a change in context is published by a PUA.

Note that in all steps, in order to store, publish, exchange, and represent context information on different type of machines, we must use an appropriate architectural model (see section 3.3). Based on the fact that SIP-SIMPLE will be used for publishing the context, PIDF (see section 3.4.3.1) was selected as the technologies to be used for modeling in this thesis. Additionally, in the printing context service we use an extended version of PWG (see section 3.4.3.2) to model printing information.

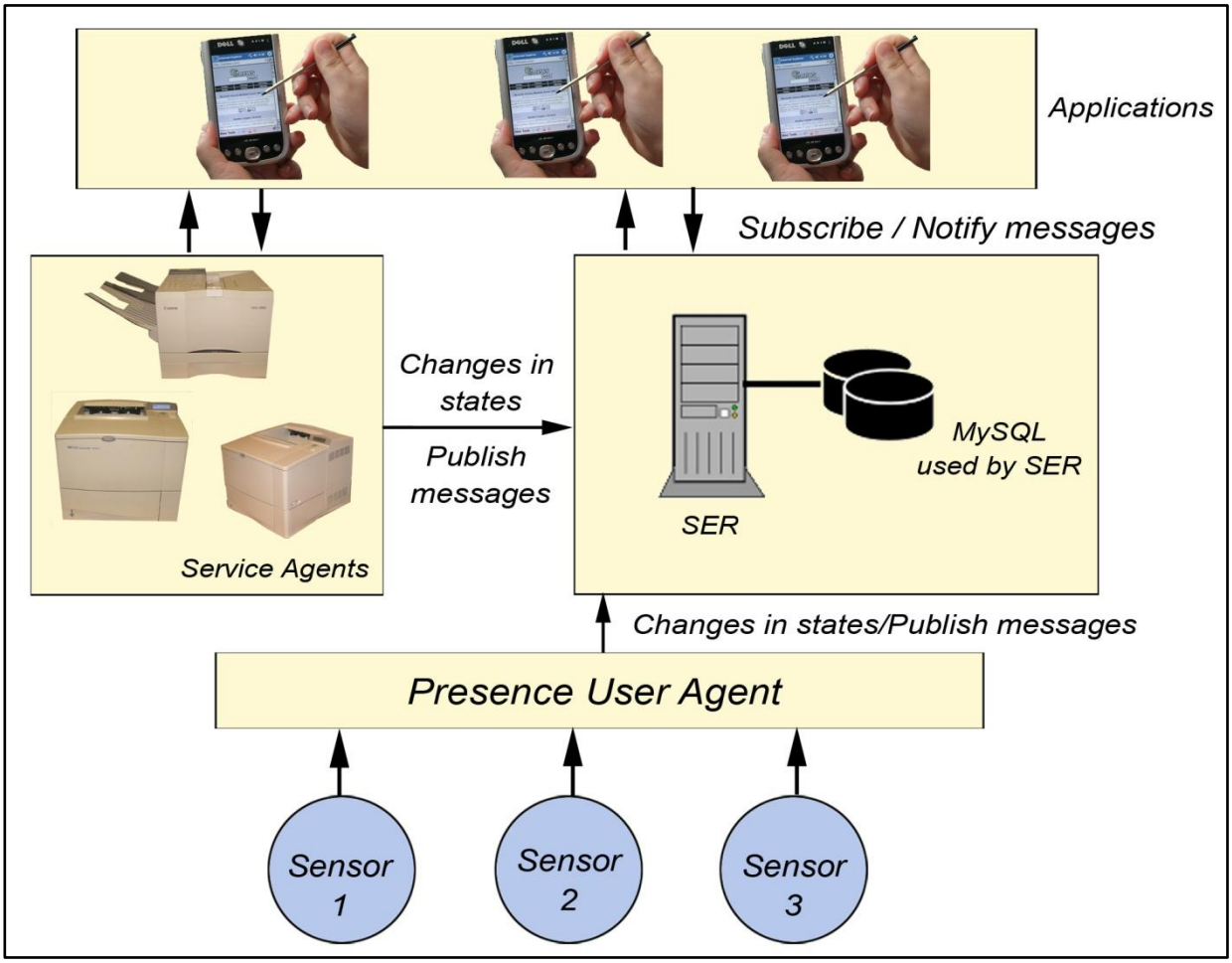

Figure 10. Proposed Architecture for this thesis

More specifically, in this thesis I am trying to develop a presence server (context aware server) can be utilized by different context applications relevant to a university campus. For example if a user (e.g. student) wants to print a document, an application on behalf of the user will ask the server for available printers around the user (using Subscribe messages). Once the server has received this information via the PUA (via the Publish messages), it will inform the user's application (using Notify messages). Therefore the user's application can, based on this context information, send the file to the preferred printer. As another example, assume that a number of students placed are on the 4th floor of the Forum building and they want to find an empty room for a group meeting and that there is an available room on the 6th floor. Their room booking application simply requests information about the number of people (room occupancy application) in different rooms, then after the server replies, the application can suggest with a list of the rooms which are **not** currently occupied by people.

5 Implementation

SER was selected as the basis of this context server in order to achieve the goals outlined earlier in this thesis. For this purpose we created a new module for SER to implement a presence server. Initially, I designed and implemented my own module (described in section 5.1), but then I learned that iptel.org had recently add a new module for SER to implement a presence server (although it only deals with the presence event), therefore I decided to modify the source code of their module, along with some other SER files, in order to enable our server to support different kinds of events (such as location) for our purposes (the details are explained in section 5.2.2).

5.1 A new SER presence user agent module

This section describes the newly developed presence usage agent module for SER. The source code is included as appendix D (notice that it is not complete as a presence agent module, and I just put them in the appendix as an example. Because when I was developing this module I noticed about the new SER Presence Agent –PA- module developed by the iptel.org and I stopped developing of my own module. The new PA module of the iptel.org explained in detail in section 5.2). Moreover, it describes how a new module can be added and loaded together with other SER modules. Notice that all SER modules are written in the C language, so we have to create our applications (modules) in C. Additionally, SER should be installed and running in a Linux system (see appendix A, for a description of how to install this on a Linux system). I have used the Ubuntu version of Linux in my thesis for the underlying operating system on the computer where this server is executed.

The context module in general must be able to handle (receive and parse) the Publish and Subscribe messages, then extract the relevant data from the header and body and store it for later use. At the same time the context module should create the Notify messages based on the context information that has been received via Publish messages, and send it to currently subscribed users. Each newly created module needs some specific code in order to be used as a SER module. Specifically, each loaded module is represented by an instance of a ‘sr_module’ structure, and there is a global variable ‘*modules’* defined in the file ‘sr_module.c’ which is the head of a linked-list of all loaded modules [61].

```
struct sr_module{
    char* path;        // This is the path used to load the module in the ser config file.
    void* handle;
    struct module_exports* exports;
    struct sr_module* next;
};
```

The ‘module_exports’ structure describes an interface that must be exported by each module. Every module must have a global variable named ‘*exports’* which is of type struct module_exports. In the following you can see the definition of this structure for our context module:

```
static cmd_export_t cmds[]={            //This structure is used to export the script
                                        //functions
    {"context",                         // null terminated command name
     Handle_Subscription,               // pointer to the corresponding function
     1,                                 //number of parameters used by the function
     0,                                 // function called to "fix" the parameters
     REQUEST_ROUTE},                    // Function flags
};
static param_export_t params[]={        //This structure is used to export the module's
                                        //parameters that can be set in ser.cfg file
    {"str_param", STR_PARAM, &str_param},
    {"int_param", INT_PARAM, &int_param},
};
struct module_exports exports = {       //This structure  is used to link all the exported
                                        //information together when defining a new
                                        //Module
    "context",                          //module name
    Handle_Subscription,                //Exported functions, can be used in ser.cfg
    Max_Publish_Expire,                 //Exported parameters, can be used in ser.cfg
    mod_init,                           //module initialization function
    0,                                  //response function
    0,                                  //destroy function
    0,                                  //on cancel function
    0                                   //per-child init function
};
```

After adding this code, our module is ready to be added to the SER modules. To do this we need to create a directory (in our case “context”) under the SER modules directory and put our module source code (context.c) in there. Inside this directory, in addition to our source code a makefile should be created, this will be used to compile the code into a dynamically loadable SER module modules. The makefile for our “context” module is shown below.

```
# Context module makefile
# WARNING: do not run this directly, it should be run by the master Makefile
include ../../Makefile.defs
auto_gen=
NAME=context.so –lpthread       /*lpthread only is needed if threads are used */
LIBS=
include ../../Makefile.modules
```

After connecting to the SER main directory we execute the following shell commands, and if there is no error, then all C files inside the context directory will be compiled and stored as a shared object with the ".so" extension (i.e., as "context.so"). Usually this will be located inside the "/usr/local/lib/ser/modules" directory. The commands are:

make clean *(this removes previously the compiled modules/binaries)*
make install *(recompiles all modules and installs them in the SER/module directory)*

Finally we should edit the ser.cfg file (it is usually located at /usr/local/etc/ser directory), and add: loadmodule "/usr/local/lib/ser/modules/context.so" to the module loading in the ser.cfg file. Additionally all exported parameters values for each module can be changed in this config file using the modparam function. Notice that modparam function accepts three parameters:

modparam("module name", "Variable name", "new value")

5.2 SER built-in Presence Agent (PA) module

Recently iptel.org has released a new module to support a presence server (the so-called PA module). However this supports only one event package (presence). Therefore this module was modified so that it can support different types of events, specifically those which are needed in this thesis (for example, location).

5.2.1 Installing and configuring

I downloaded the latest version of SER which includes the PA module from the iptel.org webpage. This file was ser-0.10.99-dev35-pa-4.2_src.tar.gz and was last modified 21-Dec-2006. Notice that this version of SER is still under developed and it is not one of the stable SER versions. After installing SER, the database for SER should be created (this version has specific tables for the PA module). A script located in source tree, in the 'scripts' directory, can be used to do this initialization of the database. Note that there are directories for several different database systems (specifically mysql and postgres). In this thesis we are using mysql, hence the relevant script to execute is "ser_mysql.sh". This script should be invoked as shown below:

/*ser_root_directory*/scripts/mysql# ./ser_mysql.sh create

This script will create the initial SER database. After this we have to specify a user name and a domain name for the system to use as a SIP URI. Other applications will use the specified host name and domain name as the server's address, when they are sending Publish or Subscribe messages. I used the host's globally routable IP address as the domain name and

the host name as the name of the user (i.e., this case it is acting as the name of the server). This data is added to the database, using the command “serctl” (which is actually is a set of command line utilities). These utilities are located in the SER source tree in the “tools/serctl” directory:

/*ser_root_directory*/tools/serctl# ./ser_domain add 130.237.15.238 Wireless

/*ser_root_directory*/tools/serctl# ./ser_user add ccsleft

/*ser_root_directory*/tools/serctl#./ser_user change ccsleft -F +sft

/*ser_root_directory*/tools/serctl#./ser_uri add 130.237.15.238 ccsleft

/*ser_root_directory*/tools/serctl# ./ser_cred add ccsleft 130.237.15.238 ccsleft *password*

This sequence of commands adds a new domain with the URI "130.237.15.238” and user "ccsleft" with password "password" to the SER database. If we use a GUI, such as MySQL client (e.g. MySQL Navigator), these values can be seen under the “domain” and “domain_attrs” tables of SER database. After adding the domain, the ser.cfg file should be edited in order to add PA support to SER (an example of a ser.cfg file supporting a PA is given in Appendix E).

5.2.2 Editing the source code of the PA module

To have the desired functions for the server for our application, the PA module and some other SER files need to be edited. As mentioned before all modules of SER have been written in C and are open source, which makes it easier for developers to change SER to meet their specific applications and requirements.

First of all, the iptel.org PA module supports only the “presence” event, and if a Publish or Subscribe massage is sent to the server with another type of event, it will reply “Unsupported event package”. Therefore, in order to handle additional event packages (location, roomA, roomB, roomC) which are needed for our application(s), the PA module source code and some other SER files have been edited. You can find the edited source code in Appendix F. Each of the edited file is described briefly below.

- **Parse_event.c and parse_event.h:** These files are located in the *ser_root_directory/*parser directory and are used by the PA module, when a subscribe/publish message is received in order to parse the event package.

- **Subscribe.c:** This file is located in the *ser_root_directory/*parser directory and is used by the PA module. It has a main function “handle_subscription” that is used to handle subscription messages when a subscribe message is received. It parses all the necessary header fields of a subscribe message. In this file I simply added the MIME

type which is used in our subscribe message for the specific event (we are using mimetype “application/pidf+xml”).

- **Watcher.c:** This file is located in the *ser_root_directory/*modules/pa directory and is used by the PA module for adding, updating, and deleting watchers (i.e., subscribed users) into/from SER database. It parses the information related to a watcher (e.g. expire value, event, status, URI, etc.), and stores it in the “watcherinfo” table. The Watcher.c file also is responsible for keeping and updating the watcher’s status (pending, active, rejected, or terminated) based on the expiration field value.

- **Publish.c:** This file is located in the *ser_root_directory/*modules/pa directory and is used by the PA module for handling received Publish messages. It is responsible for parsing and storing the required fields in the specific variables, and also creating the needed values such as Etag value (see section 2.4.1.1). The publish.c file also is responsible for checking and changing the status of a Publish messages based on expiration field values. The PA module stores event values received by subscribe messages and stores these values in the ”watcherinfo” table, but it does not do the same thing for Publish messages. Therefore, I created a new table “presentity_event” in the SER database to store the event values received by Publish messages (this can be done in either of two ways: 1. by adding the required code to the script “my_create.sql” located in the *ser_root_directory/*scripts/mysql and running the script, which will recreate all SER database tables, or 2. By executing queries via one of the Mysql client programs, such as mysqlnavigator, where the SER database is located). In the “publish.c” file I added my own code in order to parse the event field of received Publish messages and insert the results into the newly created table (e.g. “presentity_event”). In order to create the new table, the file “pa_mod.c” should also be edited (This file is the code which makes a module a valid SER module, see section 5.1 for additional information about creating new modules) and the event element has been added to the “presence_tuple_t” structure inside the presentity.h file. Later the added event should be removed from the table when the related Publish message expires. In order to do that, I added my own code to the “tuple_notec.c” file where the original SER code removes expired note field values.

  In order to edit the source code of SER modules easily, you should be familiar with the SER data structures. SER has a lot of structures which are used by its modules and without knowing them it is very difficult to add new code. You can find SER data structures with their description in the http://siprouter.onsip.org/doc/doxygen/ webpage. See Appendix F for the edited source code.

- **Notify.c:** This file is located in the *ser_root_directory/*modules/pa directory and is used by the PA module when creating Notify messages. This file is responsible for creating the different fields of a Notify message (both header and body). It uses the “pidf.c” file (located in the *ser_root_directory/*lib/presence) for parsing the different

tags of a pidf file (i.e., the body of a publish message) and also when creating the related tag values in the Notify message.

This pidf.c file has only the standard tags of pidf, as defined in [29], therefore the only tag we can use for receiving context information (from a publish message) and sending context information (to the interested subscribers) is the <note> tag (see section 3.4.3.1). To extend the PIDF tags and create our own tags, we should define a new schema and create our new tags inside the <status> tag (for more information see section 3.4.3.1). For the location event I defined a new schema for location information, such as: description (description of the specific location), room, floor, latitude, longitude, height. This defined schema is shown below:

<location>
<description> xsd:string </description> [0..1]
<room> xsd:string </room> [0..1]
<floor> xsd:integer </floor> [0..1]
<coordinates> [0..1]
<latitude> xsd:integer </latitude> [1]
<longitude> xsd:integer </longitude> [1]
<height> xsd:integer </height> [1]
</coordinates>
</location>

Notice that the schema is defined in a way that description, room, and floor tags should have at least one value, but the rest can be sent with or without a value. These specific tags are inserted in the body of Notify message only for the location event. The other event will receive a normal PIDF message (which is desired). After defining this schema, we should add our code in the file ”pidf.c” to handle these new tags. The “pidf.c” file has two main parts: 1- parsing the PIDF tags, and 2- creating the PIDF tags. After adding the code for parsing the new tags, rather than storing them in the database, instead in the “publish.c” file, these values are stored in the “presence_tuple_t” structure. Later I remove these values after sending the Notify message (notice that you must create these new elements in the “presence_tuple_t” structure inside the “presentity.h” file).

- **Presentity.c:** This file is located in the *ser_root_directory/*modules/pa directory and is used by the PA module. This file should be considered the main file of the PA module, as it uses the other files (e.g. publish.c, subscribe.c, watcher.c, notify.c, etc.) to create a presence server. In addition to the different types of data structures defined in “presentity.h”, that used in the PA module, this file is responsible for adding, updating, removing, and reading different values from/to different tables of the PA module.

This file also is responsible for sending Notify messages to Watchers (when needed) by using the function in the “notify.c” file. There is a function “process_watchers” inside the “presentity.c” file which is called periodically to check

the watchers listed in the "watcherinfo" table in order to send Notify message in different cases, such as: when a publish message with new event arrives (or when the context information of the previous event is changed), when previous publish message expires, or the subscriber's subscription has expired. New code has been added to this function in order to first check the event field of the received Publish messages, then send the Notify message only to the Watchers who have subscribed to this event (This is one of the main contribution of this master thesis).

Finally, after adding the new code to the SER modules, it should be compiled again (by using the "make install" command, see section 5.1) in the SER main directory, in order to create modules which correctly use this new code. To debug this new code quickly and to see the desired results, the debug level should be increased to 9 (in the 'ser.cfg' file) so when you run the server using the 'ser –E' command you can see more explicit information concerning the program's execution. In the source code you can use "LOG(L_ERR, "string", variable you wish to print out)", in order to print diagnostic output. Using this additional output allowed us to debug the server easily and to ensure that our new code worked as desired. However, this higher debugging level (debug=9) causes the server to output a lot of unnecessary information, this makes debugging more difficult, so it is important to hide some of this output, for example by commenting out some of the logging (e.g. "qm_free" debug messages) in the source code.

6 Evaluation

In this chapter we will try to evaluate and determine the efficiency of the proposed presence agent server (i.e., a context aware server). We will consider six different scenarios (ranging from simple to complex). The server will notify subscribed Watchers, when there is a change in the status of an event (receiving publish messages). For this purpose we have created an application in C# to simulate PUA functionality. This application simply generates different Publish messages (with different events and context information) and then sends it to our server. Also we have some Watchers, subscribed to different events and ready to receive Notify messages. In each scenario we will focus on: (a) the accuracy of the server (i.e., does the server send the correct Notify messages only to interested Watchers, based on received Publish/Subscribe messages?) and (b) The response time of the server (how long does it take for the server in each scenario to response to each of the different messages? And what are the factors that affect this bottleneck?). This chapter concludes with a discussion about some of the vulnerabilities of the proposed server.

6.1 One Watcher only subscribing

In this scenario only one Watcher has subscribed to the server. In this simple case we try to show the all messages between the server and Watcher. As each of the messages is time stamped, we can calculate the actual subscription time-out. We adjust the value of the 'Expire' field in the Subscribe message (specifically using the values: 5, 10, 20, and 30 seconds). We will measure the time from the subscription being sent until the Watcher receives the Notify message (which means that the subscription has expired) and compare it with the value of 'Expire' field to see how close these are to each other.

The messages flow is shown in the Figure 11. Here there are 6 different messages. The Subscribe message (M1) is the first message sent from the Watcher to the server in order to subscribe to an event which is being monitored by the server (in this example scenario the 'Event' field value is not important). Then the server replies with OK (M2) and Notify (M3) messages to indicate that it received the message M1 and to send the current event status using a Notify message (in this scenario this is an empty notification because there was no previous Publish message). Next the Watcher sends an OK message (M4) to the server to indicate that it received the notification. Later when the subscription message has expired, the server sends a Notify message (M5) to inform the Watcher that its subscription has expired. The final message is an OK (M6) message from the Watcher to the server to indicate that it has received the notification of the expired subscription.

In this scenario we are using a PDA[2] which can create and send different subscription messages with different 'Expire' values. On the server side, SER is running on a Dell "OptiPlex GX620" computer equipped with a 2.80 GHz "Intel Pentium D" processor and 2.0

[2] The PDA used was an HP iPAQ model 5550. This PDA has an IEEE 802.11b WLAN interface. The PDA used this WLAN interface to communicate via a 'Cisco Aironet 1000, AIR – AP1020- K9' WLAN access point to KTH's wired network. The server is attached to this wired network.

GB of memory. The SER server is operating and ready to receive SIP messages. Wireshark [70], a network packet capture and decode application, is running on the same computer as the SER server. Wireshark has been configured to capture all UDP messages destined to the UDP port number '5060' (therefore in this scenario it captures only the relevant messages to/from the server). You can see the 'Wireshark' output in the table 4. The table has been divided into 7 sections (each section concerns a different expiration value: 5, 10, 20, and 30 seconds). Each section of the table has 6 columns, which indicates the transfer of each of the messages (ranging from M1 to M6) between the Watcher (note this has the IP address130.237.238.63) and the server (with the IP address 130.237.15.238). The time column indicates the time stamp for each message relative to the start of the test. In our analysis we will on the data in this column.

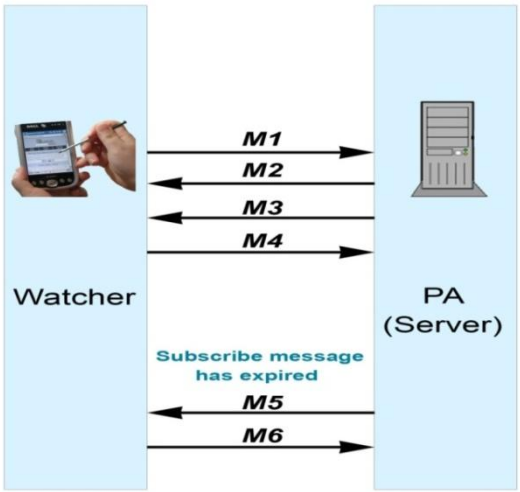

Figure 11. M1: Subscribe, M2: OK, M3: Notify, M4: OK,

M5: Notify (for Expiration of Subscribe messages), M6: OK

Carefully examining each section of table 4 proves that the correct messages have been sent between the server and Watcher. Therefore in this scenario we can see that the server functions correctly. However, we see that there are two M3 messages in each subtest. This occurs because the server did not receive the OK message from the Watcher (note that the delays between the various M3 messages are a constant compiled into SER). To prove this, we disabled the sending of the OK message from the Watcher side (i.e., it has been configured so that when it receives the Notify message it will **not** send an OK message) to see whether the server will send the Notify message continuously or not. The result is shown in the table 5, and as you can see the server sends Notify messages continuously if it does not receive an OK message, this continues until the Subscribe message has expired (however, to limited the space needed we have only included five of these Notify messages). Examining the time column, you can easily see that the server increases the delay between each of the subsequent Notify messages. This feature has some advantages and disadvantages. It can be positive, that the server will send the Notify message until it receives the OK message from the Watcher - this ensures that the Watcher received the server's Notify message. On the other hand, if the Watcher receives the Notify message, but cannot send the OK message to the server, the server will send a lot of Notify messages through the network which is not desirable. Because of the back off, the rate at which the subsequent copies of the Notify message are set decreases - which does help to reduce the network traffic.

Table 4. Wireshark output for the proposed scenario

No.	Time (seconds)	Source	Destination	Protocol	Info
M1	**5.912099**	130.237.238.63	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238
M2	5.919079	130.237.15.238	130.237.238.63	SIP	Status: 200 OK
M3	5.920027	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M3	6.386667	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M4	6.456216	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
M5	**11.63515**	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M6	11.69623	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
Expiration Value In Subscription Message (M1) = 5 second					
M1	**31.46537**	130.237.238.63	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238
M2	31.46847	130.237.15.238	130.237.238.63	SIP	Status: 200 OK
M3	31.4688	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M3	31.94582	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M4	32.01809	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
M5	**41.63414**	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M6	41.69486	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
Expiration Value In Subscription Message (M1) = 10 second					
M1	**62.61695**	130.237.238.63	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238
M2	62.62152	130.237.15.238	130.237.238.63	SIP	Status: 200 OK
M3	62.62263	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M3	63.07269	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M4	63.14344	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
M5	**83.63271**	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M6	83.69472	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
Expiration Value In Subscription Message (M1) = 20 second					
M1	**91.1972**	130.237.238.63	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238
M2	91.21741	130.237.15.238	130.237.238.63	SIP	Status: 200 OK
M3	91.21756	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M3	91.69983	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M4	91.76801	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
M5	**121.6366**	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M6	121.6989	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
Expiration Value In Subscription Message (M1) = 30 second					

Table 5. Wireshark output, when the Watcher does not reply with OK message (note that only five copies of the message M3 are shown, these messages continue to be sent until the subscription expires)

No.	Time (seconds)	Source	Destination	Protocol	Info
M1	8.878187	130.237.238.63	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238
M2	8.883079	130.237.15.238	130.237.238.63	SIP	Status: 200 OK
M3	8.884546	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M3	9.363678	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M3	10.36364	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M3	12.36361	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M3	16.36348	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63

By analyzing the time column in table 4, we learn that one unit in the field of 'Expire' value from the subscription means approximately 1 second at the server (note that the measurement repeats until the expiration value reaches 90 seconds, but due to space limitations the results are shown only up to 30 seconds). For example, if you look at the second section of this table (which shows the measurements when the expiration value was 10 seconds), the M1 message (the Subscribe message) was sent at 31.46 to the server, while the M5 message (the Notify to the Watcher of the expiration of its subscription) was send at 41.63, indicating that the subscription was ~10.17 seconds.

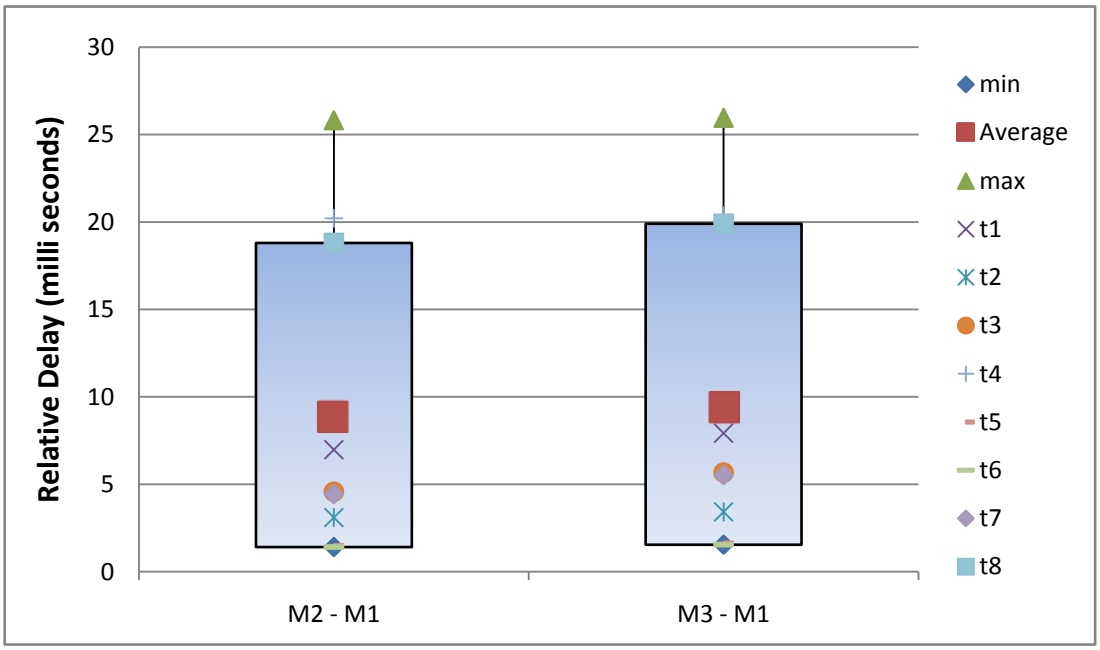

Figure 12. The relative delay of M2 and M3 to M1 from table 4. Measurements t1 .. t8 are the different M2 and M3 message time stamps (from the time column) subtracted from the relevant M1. Also min, max, and the average time are calculated for both M3-M1 and M2-M1

Figure 12 shows the relative delay between the two messages: the delay of the OK message (M2) and the delay of the Notify message (M3) from the subscription message (M1). In the figure 12, there are ten samples in each column (for the expiration values: 5, 10, 20, …, 90 seconds) which represented by t1, …, t8. Additionally the minimum, maximum, and average of these ten samples are calculated and shown. As you can see in the first column (M2-M1) ranges from ~1.4 until ~25 ms , with an average of ~9 ms, whereas the second column (M3-M1) ranges from ~1.5 until ~26 ms, with an average of ~9 ms. Therefore in this scenario (with one Watcher), the Watcher will receive the reply (both the OK and the immediate Notify message) within in approximately in 10 ms and there is less than 1 ms delay between the sending of the OK and the Notify messages to the Watcher. We can summarize the results of this section as below:

- The time requested for the subscription is in fact the duration of the subscription to within ~10ms.
- The processing of individual subscription requests takes less than ~25ms.
- The initial Notify message follows the OK, by less than 1ms.

6.2 One Watcher and One Publisher (for a single type of event)

This scenario includes three parts: (a) In stage 1, only one Watcher has subscribed to the server (for the location event), and we send the Publish message from a PUA (in this test using an application which simulates PUA functionality) to the server. Therefore the server must notify the Watcher when it receives the Publish messages. (b) In stage 2 of this test, we change the context information (simulating a change in the user's location) in the Publish messages (for the same event), and send this updated information to the server (by using the Publish's Modify message, see chapter 5), to see whether the server will correctly notify the Watcher. (c) In the last stage we send only Publish messages with a short interval between them to see whether the server can handle all of them or not. We try to show all the messages between the server, PUA, and Watcher with along their time stamps. We also show the published message time-out. We set the value of the 'Expire' field in the Publish message to '20', thus we will measure the time from the Publish (from the message is sent by the PUA to the server) until the Watcher receives the Notify message (from the server, which means the Publish message has expired) and compare it with the value of 'Expire' field to see how close they are to each other.

The message flow for this part of the scenario is shown in the Figure 13, consisting of 10 different messages. The Subscribe message (M1) is the first message from Watcher (it has the IP address 130.237.238.63) to the server (with the IP address of 130.237.15.238) in order to subscribe to the 'location' event from the server (in this test we set a high value for the 'Expire' field in the Subscribe messages and do not show the expiration of subscription in the messages flow). Next the server replies with OK (M2) and Notify (M3) messages to indicate

that it received the M1 and to send the current event status with this Notify message (initially this is an empty notification, because we did not yet send any Publish message). In response the Watcher sends an OK message (M4) to the server to indicate that it received the notification. After the Watcher has subscribed, the PUA (with the IP address of 130.237.15.196) sends a Publish message (M5) with the 'location' value for the 'Event' field and a value of '20' for the 'Expire' field in order to inform the server that there is a change in the state of the entity's location. After receiving this message the server first replies to the PUA with an OK (M6) to indicate that it has received the Publish message, then the server sends a Notify message (M7) to the Watcher to inform it of this update in the status of the watched entity's location. To this notification the Watcher replies with an OK (M8). Later, when the Publish message has expired, the server sends a Notify message (M9) only to the Watcher to inform it about the expiration of the previously received Publish message. The final message is an OK (M10) message from the Watcher to the server to indicate that it has received the notification.

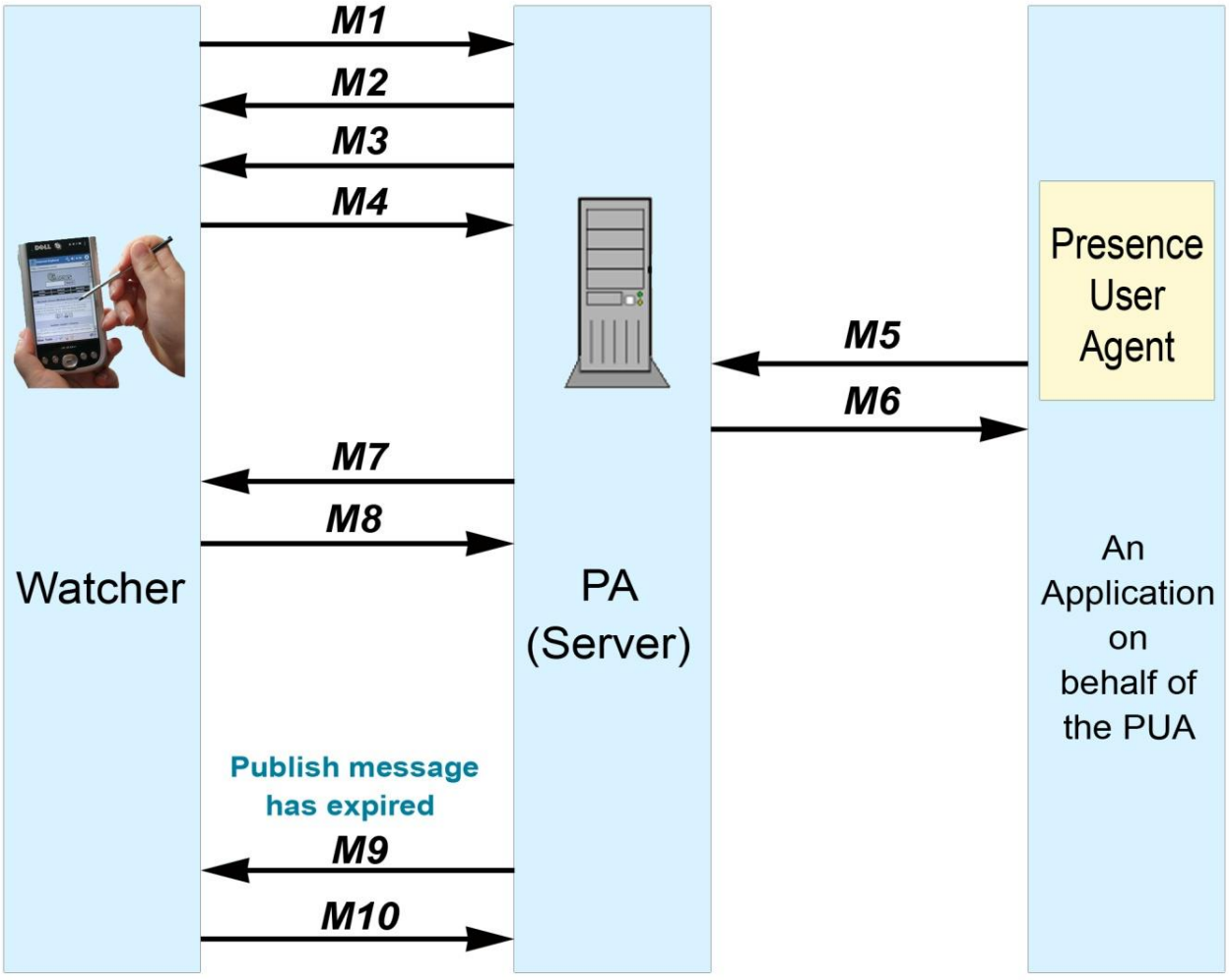

Figure 13. M1: Subscribe, M2: OK, M3: Notify, M4 : OK, M5: Publish for updates M6: OK, M7: Notify, M8: OK, M9: Notify (for expiration of the Publish), M10: OK

Looking at the 'time' column in table 6, we can again see that one unit in the field of the 'Expire' value from the Publish message (M5) means approximately 1 second at the server. If you calculate the difference in time from the M5 (the Publish message) until M9 (the notification for M5's expiration), the result is '31.98 – 9.93 =22.05' seconds; this is comparable with the Publish expiration value (20 seconds). Thus we see that the expiration behavior is as expected.

Looking at the messages among the Watcher, server, and PUA; we can see that the correct messages have been sent and that they have been sent in the correct order. Therefore in this scenario the server functions correctly. However, as in the first scenario the M3 message is repeated. As in the first scenario, there is not a third instance of the M3 message; because the M4 OK message acknowledges one of the M3 messages (note that we do not know which one is being acknowledged - as the messages are identical). Thus we do not know if the delay was ~68ms or ~512ms. For a discussion of this- see section 6.1.

Table 6. Wireshark output for One Watcher and One Publisher (stage 1)

No.	Time	Source	Destination	Protocol	Info
M1	**2.340096**	130.237.238.63	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238
M2	2.34756	130.237.15.238	130.237.238.63	SIP	Status: 200 OK
M3	2.348818	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M3	**2.793675**	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M4	2.861404	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
M5	**9.935065**	130.237.15.196	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238
M6	9.938114	130.237.15.238	130.237.15.196	SIP	Status: 200 OK
M7	**9.982441**	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M8	10.044201	130.237.238.63	130.237.15.238	SIP	Status: 200 OK
M9	**31.982346**	130.237.15.238	130.237.238.63	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.63
M10	32.042451	130.237.238.63	130.237.15.238	SIP	Status: 200 OK

In the second stage of this scenario, we assume that the location of the user is changing, thus we can examine how the server functions with regard to these location updates. As noted earlier we do not actually have the user move, but rather generate a series of messages as if they moved. For this purpose, every 5 seconds the PUA sends a Publish message (in the same conversation using the Modify version of the Publish message) with the new location of the user (this is the updated context information). In order to do this, the PUA extracts the location name from an array and sends an updated Publish message to the server. The results are shown in the table 7.

Note that an additional column has been added to the Wireshark output 'Location' to show the context information in the different Publish and Notify messages (This information was manually copied from the packet). This column shows that whenever the PUA sends a

new Publish message with a new location, the server sends the updated context information to the Watcher in the correct format. Looking at the timing column indicates that the Watcher is notified of the new context information within 0.3 – 1.3 second (this is the server's response time with a single Watcher). In the next paragraphs, we will see that this time delay is due to a parameter setting in the server.

Table 7. Wireshark output for One Watcher and One Publisher (stage 2)

No.	Time	Source	Destination	Protocol	Info	Location
M1	42.42801	130.237.238.112	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238	-
M2	42.4486	130.237.15.238	130.237.238.112	SIP	Status: 200 OK	-
M3	42.44986	130.237.15.238	130.237.238.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.112	-
M3	42.91822	130.237.15.238	130.237.238.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.112	-
M4	42.99693	130.237.238.112	130.237.15.238	SIP	Status: 200 OK	-
M5	52.37989	130.237.238.87	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	lab
M6	52.38179	130.237.15.238	130.237.238.87	SIP	Status: 200 OK	-
M7	53.47874	130.237.15.238	130.237.238.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.112	lab
M8	53.55069	130.237.238.112	130.237.15.238	SIP	Status: 200 OK	-
M5	57.14277	130.237.238.87	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	library
M6	57.1486	130.237.15.238	130.237.238.87	SIP	Status: 200 OK	-
M7	57.47877	130.237.15.238	130.237.238.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.112	library
M8	57.57478	130.237.238.112	130.237.15.238	SIP	Status: 200 OK	-
M5	62.14123	130.237.238.87	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	cafeteria
M6	62.14326	130.237.15.238	130.237.238.87	SIP	Status: 200 OK	-
M7	63.47838	130.237.15.238	130.237.238.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.112	cafeteria
M8	63.56208	130.237.238.112	130.237.15.238	SIP	Status: 200 OK	-
M5	67.1442	130.237.238.87	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Wireless
M6	67.15007	130.237.15.238	130.237.238.87	SIP	Status: 200 OK	-
M7	67.47832	130.237.15.238	130.237.238.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.112	Wireless
M8	67.55607	130.237.238.112	130.237.15.238	SIP	Status: 200 OK	-
M5	72.14077	130.237.238.87	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	lab
M6	72.14289	130.237.15.238	130.237.238.87	SIP	Status: 200 OK	-
M7	73.48214	130.237.15.238	130.237.238.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.238.112	lab
M8	73.55427	130.237.238.112	130.237.15.238	SIP	Status: 200 OK	-

In order to find out that how long we should wait between two Publish messages, so that our server can handle them in a correct way, I decreased the time interval between Publish messages. This revealed that the minimum interval between two Publish messages is 1.00 second. If Publish messages arrive more frequently than one per second, then the server dos not correctly handle all of the Publish messages. Further experiments showed that the results get worse as the interval between publish messages decreases. Table 8 shows the output of Wireshark (from the server's point of view) where the Publish messages are sent with a 0.5 second interval between them. As can be seen in table 8, the server does not correctly handle some of the Publish messages (marked in red in the table), while it correctly acknowledges the PUA's Publish message, the Watcher is not notified of this change. On the other hand, as we saw in the previous scenario and stage 1, the server responds within 4-5 ms. So it seems that the problem is not with the server processing the PUBLISH messages (see end of this section); but rather the reason for this issue (the server is not able to accepted PUA location updates quickly) is that there is a rate limit for the NOTIFY messages (about one per second) in the ser.cfg file where the parameter of the PA module is set (see appendix E). Thus it would appear that the notifications to the Watcher are rate limited to a maximum of once per second. However, when the server does finally send out a Notify message, the Watcher always receives the correct (and the latest) context information. It is important to decide for all of the expected applications, how often the server should test if something has changed and if so to send a Notify message to the Watcher. For example, if for a 'location based reminder' application (such as that of [64]) it is not necessary to inform Watchers very quickly (because the user is not moving so quickly- walking speed is about 2m/s) , then a longer delay in sending NOTIFY messages is acceptable. In any case, how often to send a Notify message can be configured in ser.cfg, by setting the module's parameters (appendix E, shows: modparam("pa", "timer_interval", 2) which results in a check for changes once every 2 seconds - which results in an average time between a change occurring and the NOTIFY being set of 1 second). One problem which should be noted is that this single parameter determines the maximum rate at which the systems checks for changes for all events, thus it impacts **all** applications which use this server. So if even one application needs faster event notifications, then the parameter would need to be changed and the SER server restarted.

Table 8. Wireshark output with 0.5 second time interval between Publish messages

No.	Time	Source	Destination	Protocol	Info	Location
M1	54.754345	130.237.239.213	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238	-
M2	54.775679	130.237.15.238	130.237.239.213	SIP	Status: 200 OK	-
M3	54.776573	130.237.15.238	130.237.239.213	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.213	-
M4	55.295776	130.237.239.213	130.237.15.238	SIP	Status: 200 OK	-
M5	58.788145	130.237.239.12	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Lab
M6	58.793403	130.237.15.238	130.237.239.12	SIP	Status: 200 OK	-
M5	59.088904	130.237.239.12	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Library
M6	59.094913	130.237.15.238	130.237.239.12	SIP	Status: 200 OK	-
M7	59.160219	130.237.15.238	130.237.239.213	SIP/XML	Request: NOTIFY	Library

					sip:Sub1@130.237.239.213	
M8	59.230333	130.237.239.213	130.237.15.238	SIP	Status: 200 OK	-
M5	59.594796	130.237.239.12	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Cafeteria
M6	59.599527	130.237.15.238	130.237.239.12	SIP	Status: 200 OK	-
M5	60.093641	130.237.239.12	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Wireless
M6	60.095752	130.237.15.238	130.237.239.12	SIP	Status: 200 OK	-
M7	60.16002	130.237.15.238	130.237.239.213	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.213	Wireless
M8	60.237627	130.237.239.213	130.237.15.238	SIP	Status: 200 OK	-
M5	60.590623	130.237.239.12	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Lab
M6	60.602115	130.237.15.238	130.237.239.12	SIP	Status: 200 OK	-
M5	61.092723	130.237.239.12	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Library
M6	61.094842	130.237.15.238	130.237.239.12	SIP	Status: 200 OK	-
M7	61.159923	130.237.15.238	130.237.239.213	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.213	Library
M8	61.238517	130.237.239.213	130.237.15.238	SIP	Status: 200 OK	-
M5	61.59331	130.237.239.12	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Cafeteria
M6	61.615011	130.237.15.238	130.237.239.12	SIP	Status: 200 OK	-
M5	62.093741	130.237.239.12	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Wireless
M6	62.095971	130.237.15.238	130.237.239.12	SIP	Status: 200 OK	-
M7	62.163894	130.237.15.238	130.237.239.213	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.213	Wireless
M8	62.2427	130.237.239.213	130.237.15.238	SIP	Status: 200 OK	-

Figures 14, 15, and 16 show the server's response times in terms of two different messages: M6 (the OK reply to the PUA) and M7 (the Notify messages to the Watcher). There are two interesting aspects of these plots: (a) the server has two different ranges of response times. The delay between M5 and M6 is around 0.34 second, while the delay between M5 and M7 is around 1.3 second. This later high delay is because the server uses a timer to determine when to check for changes in the event status (i.e. when it processes publish messages), and based on when the Publish message was received. As we have set the parameter such that the server will check every two seconds, thus if publish events arrive randomly distributed the expected time before checking will be one second. In addition to this one second we need to add the base response time of 0.34 seconds (which was measured with the M6-M5 measurement); therefore one would expect that the M7-M5 delay would be 1.34 seconds). And (b) if the 'Expire' value in Publish messages is set to higher value, and these Publish messages have been received frequently by server, the server keeps these Publish messages (even though it only uses the latest one). Therefore if there is large number of the Publish messages in the server's database, when a new Publish message receives, it takes more time (compared with the case when there was no Publish message in the server's

database) for the server to handle this new Publish message. To address this issue, I set a small value (i.e., 5 seconds) for the maximum validity of the Publish messages in the ser.cfg file (see Appendix E), therefore the server removes old records , which reduces the average response time needed to process each PUBLISH message. This comes at the cost of not keeping information about where the entity has been for longer than 5 seconds. This might negatively impact applications that would like to query the database for the history of an entity's location values. However, it does mean that the database is not keeping an extensive history, which might improve the system with regard to maintaining the user's privacy & integrity.

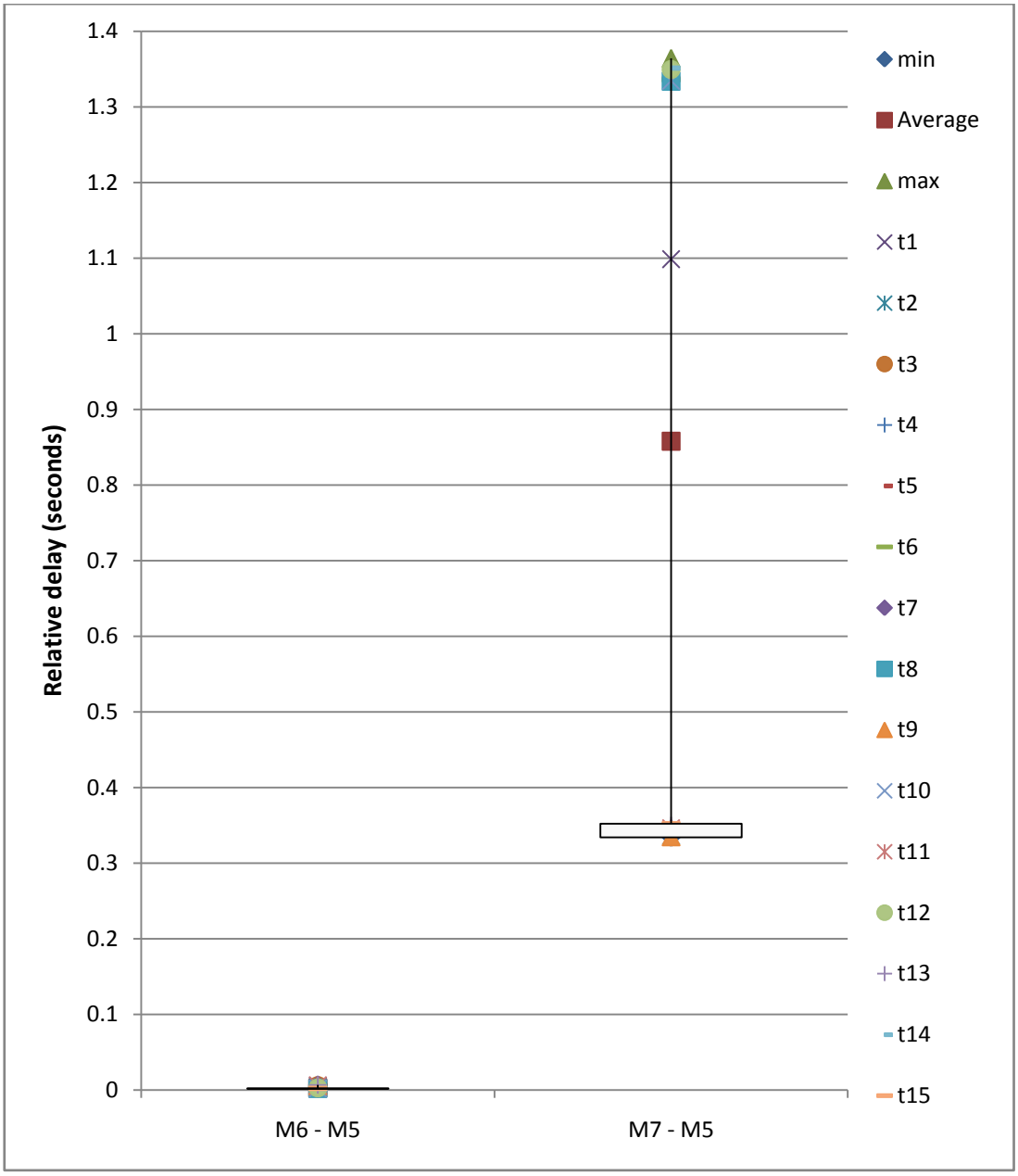

Figure 14. The relative delay (in seconds) between M6-M5, and M7-M5

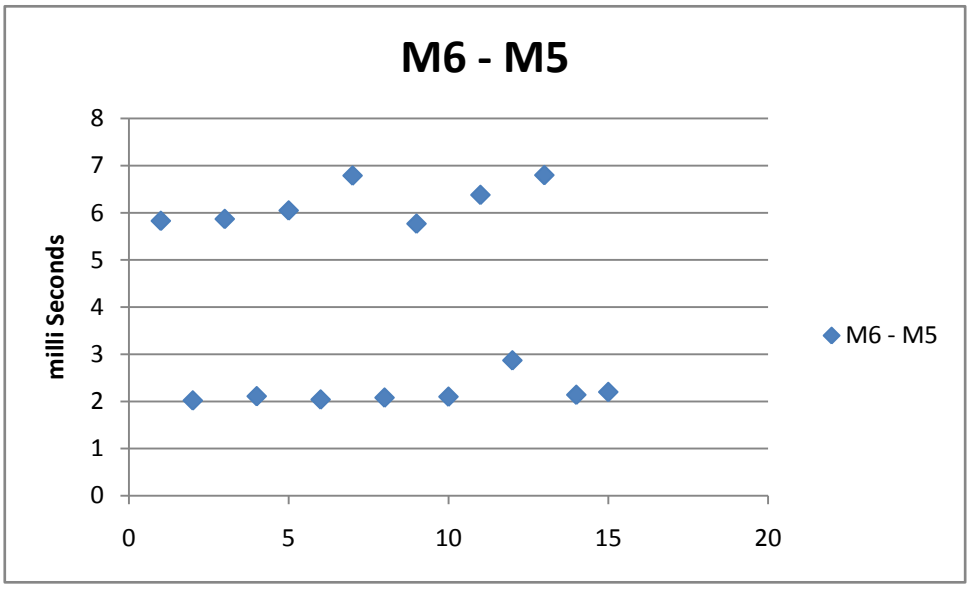

Figure 15. The relative delay (in seconds) between M6-M5

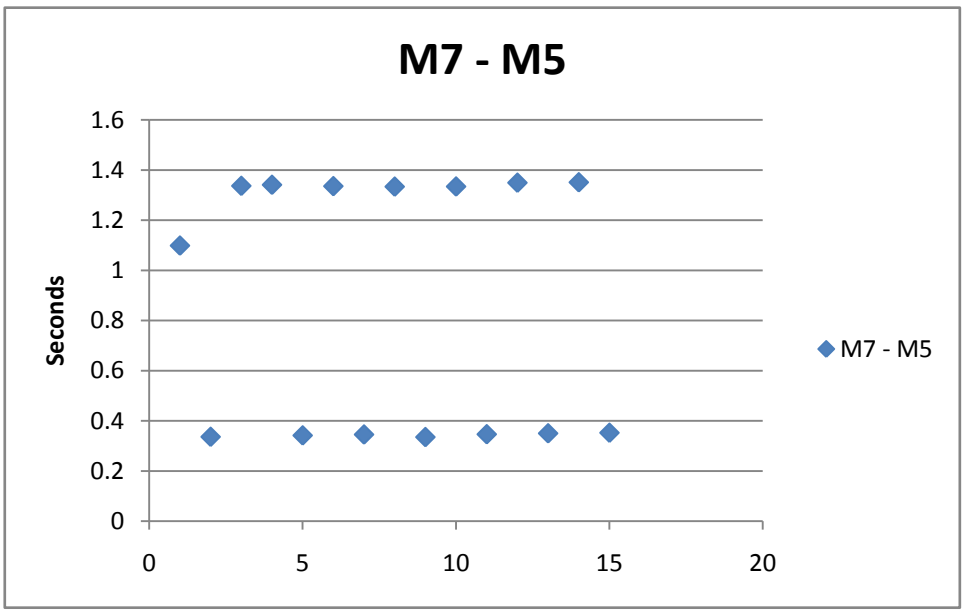

Figure 16. The relative delay (in seconds) between M7-M5

The last test in this section was performed to examine how the server handles Publish messages. For this test I used four PDAs (Pub1, Pub2, Pub3, and Pub4). Each PDA sent 50 Publish messages with only 60 ms (PDAs cannot handle less that 60 ms interval between two Publish messages) interval between them (leading to in total 200 messages). This high rate of message is used to see if the server can handle all of these messages correctly, by replying

with an OK message and updating its database with the updated information (from the Publish messages). Table 9 shows some of these results (due to limited space only some of the messages are shown) for this test. Analyzing the time column shows that, although some of the Publish messages were received by the server and processed with less than 5ms (the red lines), the sever correctly handles all of the messages and replied with an OK messages to the correct source PUA. The database was checked to see if the server only replied with an OK message or if it also correctly updates its database in this short time interval. The result shows that the server inserted the information of all 200 Publish messages in its database. Therefore the server not only can receive and reply to a large number of Publish messages with a short time interval between them (less than 5ms), but it can update its database correctly within this short time. Thus we know that the server can handle at least this rate of updates for at least in 12.64 seconds (considering 60ms time interval between Publish messages (200*60ms= 12 seconds)).

Table 9. Wireshark output for Publish messages from four PDAs with 60ms interval between these messages

No.	Time	Source	Destination	Protocol	Info	Messages
1	0	130.237.238.241	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub1
2	0.005118	130.237.15.238	130.237.238.241	SIP	Status: 200 OK	OK1
3	0.034198	130.237.238.217	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub2
4	0.039616	130.237.15.238	130.237.238.217	SIP	Status: 200 OK	OK2
5	0.063251	130.237.239.110	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub3
6	0.071352	130.237.15.238	130.237.239.110	SIP	Status: 200 OK	OK3
7	0.142803	130.237.238.90	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub4
8	0.148255	130.237.15.238	130.237.238.90	SIP	Status: 200 OK	OK4
9	0.267858	130.237.238.241	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub1
10	0.272794	130.237.15.238	130.237.238.241	SIP	Status: 200 OK	OK1
11	0.296936	130.237.238.217	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub2
12	0.302923	130.237.15.238	130.237.238.217	SIP	Status: 200 OK	OK2
13	0.3249	130.237.239.110	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub3
14	0.335577	130.237.15.238	130.237.239.110	SIP	Status: 200 OK	OK3
15	0.45041	130.237.238.90	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub4
16	0.455729	130.237.15.238	130.237.238.90	SIP	Status: 200 OK	OK4
17	0.539669	130.237.238.241	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub1
18	0.554659	130.237.238.217	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub2
19	0.608957	130.237.15.238	130.237.238.241	SIP	Status: 200 OK	OK1

20	0.609614	130.237.239.110	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub3
21	0.623876	130.237.15.238	130.237.239.110	SIP	Status: 200 OK	OK3
22	0.706212	130.237.238.90	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub4
23	0.711548	130.237.15.238	130.237.238.90	SIP	Status: 200 OK	OK4
24	0.722705	130.237.15.238	130.237.238.217	SIP	Status: 200 OK	OK2
33	1.076967	130.237.238.217	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub2
34	1.081202	130.237.238.241	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub1
35	1.082676	130.237.15.238	130.237.238.217	SIP	Status: 200 OK	OK2
36	1.092889	130.237.15.238	130.237.238.241	SIP	Status: 200 OK	OK1
37	1.109396	130.237.239.110	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub3
38	1.114571	130.237.15.238	130.237.239.110	SIP	Status: 200 OK	OK3
39	1.266408	130.237.238.90	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub4
40	1.271507	130.237.15.238	130.237.238.90	SIP	Status: 200 OK	OK4
59	**1.889278**	130.237.239.110	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub3
60	**1.89378**	130.237.238.241	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub1
61	1.900552	130.237.15.238	130.237.239.110	SIP	Status: 200 OK	OK3
62	1.911094	130.237.15.238	130.237.238.241	SIP	Status: 200 OK	OK1
79	**2.673935**	130.237.239.110	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub3
80	**2.675357**	130.237.238.217	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub2
81	**2.681664**	130.237.238.90	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Pub4
82	2.6825	130.237.15.238	130.237.239.110	SIP	Status: 200 OK	OK3
83	2.694111	130.237.15.238	130.237.238.90	SIP	Status: 200 OK	OK4
84	2.699046	130.237.15.238	130.237.238.217	SIP	Status: 200 OK	OK2

6.3 Multiple Watchers subscribed to one event (location)

In this scenario there are three Watchers (Sub1, Sub2, and Sub3), subscribed for a 'Location' event. As in the previous scenario, every 5 seconds an application on behalf of the PUA sends an updated Publish message with new context information (emulating a new location of the user). This scenario (a) measures the server's response time as in the previous scenario in order to see whether it changes or not, and (b) as there are now multiple subscribers we examine in which order the server sends the Notify messages to the Watchers (e.g. first Sub1, then Sub2, etc.) and if this order changes or not. The message flow is shown in Figure 17, which is similar to the previous scenario, except now there are **three** Watchers.

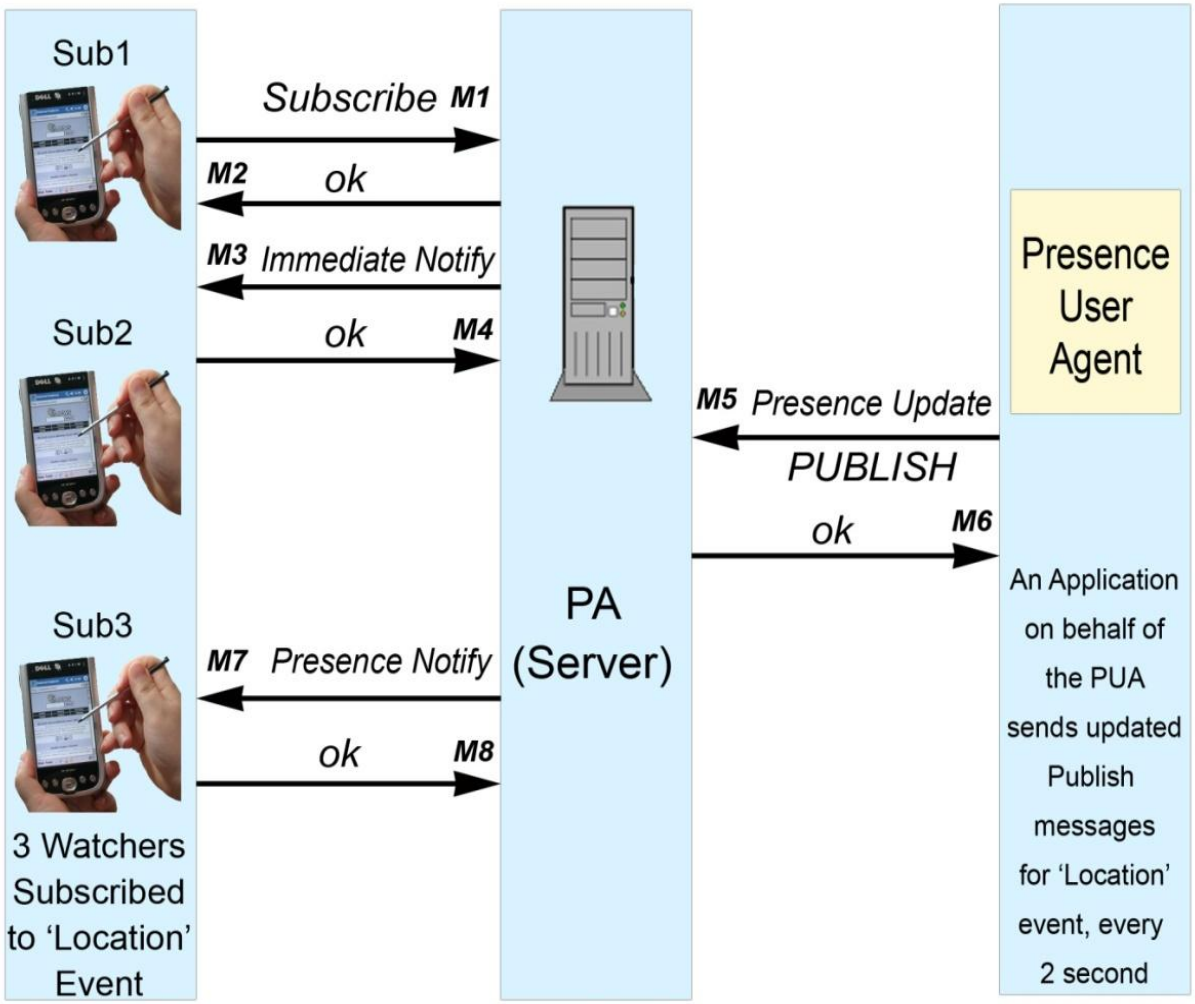

Figure 17. Three Watchers (Sub1, Sub2, and Sub3) subscribed for a 'Location' Event; the PUA periodically sends updated Publish messages with this 'Location' Event to the Server

The 'info' column of table 10 indicates that Sub1, Sub2, and Sub3 subscribe to the server in sequence. The experimental results show that when the server sends the Notify message, to the last Watcher (to subscribe) first (i.e. the order of the notification is Sub3, Sub2, and then Sub1). Examining the server's database shows that when a Watcher subscribes, the server put this subscription in the table as a new record and when the server wants to send a Notify message it reads from this table moving from the bottom to the top and checks whether to send the Notify message or not (i.e., based upon if this Watcher has indicated that it is interested in this event or not).

Because all three Watchers subscribed for the 'Location' event, the server should send each one of them the Notify message, after it receives the Publish message. Table 10 clearly shows that the server in fact sends all of the Watchers their own copy of the Notify message, following the receipt of the Publish. In addition, the table (and the code) establishes that the order is in fact constant and is in the reverse order of the subscriptions.

Table 10. The Wireshark output for the proposed scenario

No.	Time	Source	Destination	Protocol	Info	Location
M1	4.46917	130.237.239.112	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238	
M2	4.47659	130.237.15.238	130.237.239.112	SIP	Status: 200 OK	
M3	4.957622	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	
M4	5.027342	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M1	6.834702	130.237.238.96	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238	
M2	6.835747	130.237.15.238	130.237.238.96	SIP	Status: 200 OK	
M3	7.333555	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	
M4	7.403255	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M1	9.761191	130.237.238.242	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238	
M2	9.762995	130.237.15.238	130.237.238.242	SIP	Status: 200 OK	
M3	10.205524	130.237.15.238	130.237.238.242	SIP/XML	Request: NOTIFY sip:Sub3@130.237.238.242	
M4	10.271046	130.237.238.242	130.237.15.238	SIP	Status: 200 OK	
M5	23.854923	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Lab
M6	24.078284	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	24.205937	130.237.15.238	130.237.238.242	SIP/XML	Request: NOTIFY sip:Sub3@130.237.238.242	Lab
M7	24.206816	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	Lab
M7	24.207593	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	Lab
M8	24.26898	130.237.238.242	130.237.15.238	SIP	Status: 200 OK	
M8	24.270271	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M8	24.272057	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M5	28.859346	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Library
M6	28.861749	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	30.205696	130.237.15.238	130.237.238.242	SIP/XML	Request: NOTIFY sip:Sub3@130.237.238.242	Library
M7	30.206563	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	Library
M7	30.207384	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	Library
M8	30.273796	130.237.238.242	130.237.15.238	SIP	Status: 200 OK	
M8	30.275953	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M8	30.277476	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M5	33.858414	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Cafeteria

M6	33.864657	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	34.209574	130.237.15.238	130.237.238.242	SIP/XML	Request: NOTIFY sip:Sub3@130.237.238.242	Cafeteria
M7	34.210602	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	Cafeteria
M7	34.211486	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	Cafeteria
M8	34.278748	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M8	34.281719	130.237.238.242	130.237.15.238	SIP	Status: 200 OK	
M8	34.283723	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M5	38.854171	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Wireless
M6	38.857114	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	40.20946	130.237.15.238	130.237.238.242	SIP/XML	Request: NOTIFY sip:Sub3@130.237.238.242	Wireless
M7	40.210348	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	Wireless
M7	40.21115	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	Wireless
M8	40.283923	130.237.238.242	130.237.15.238	SIP	Status: 200 OK	
M8	40.284472	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M8	40.385843	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M5	43.854626	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Lab
M6	43.861162	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	44.209346	130.237.15.238	130.237.238.242	SIP/XML	Request: NOTIFY sip:Sub3@130.237.238.242	Lab
M7	44.210239	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	Lab
M7	44.211042	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	Lab
M8	44.273548	130.237.238.242	130.237.15.238	SIP	Status: 200 OK	
M8	44.274531	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M8	44.282156	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	

Examining the server's response time (the time from getting the Publish message until sending the Notify message, (i.e. M5-M7)) is the final analysis of data from this scenario. Figures 18, 19, 20, and 21 show a plot of these delays for Sub3, Sub2, and Sub1. Comparing the results with table 8 from the previous scenario and table 10 in this scenario; we see that the first Watcher (Sub3) receives the Notify message with almost exactly the same distribution of delays as the single Watcher in the previous scenario. Therefore increasing the number of the Watchers does not increase the server response time for the last Watcher. Moreover, the delay until the Notify message is received by the second and the third Watchers (here Sub2 and Sub1) shows that each Watcher receives the Notify messages approximately 1ms later than the first Notify. Note that as was the case in section 6.2, the successive delays roughly alternate between a roughly minimal value and a roughly maximal value.

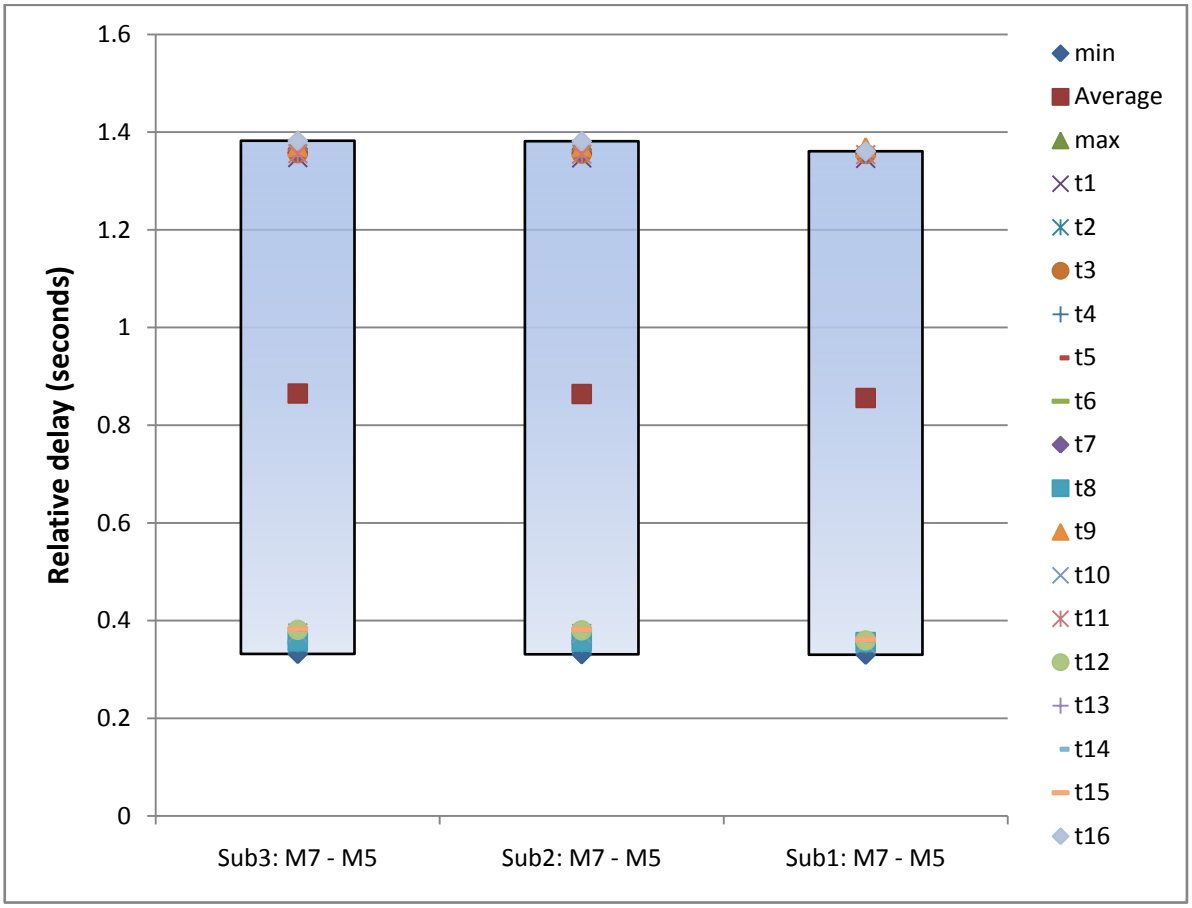

Figure 18. The relative delay (in seconds) between M7-M5 for Sub1, Sub2 and Sub3; results are shown for the 15 sets of measurements (t1 .. t16)

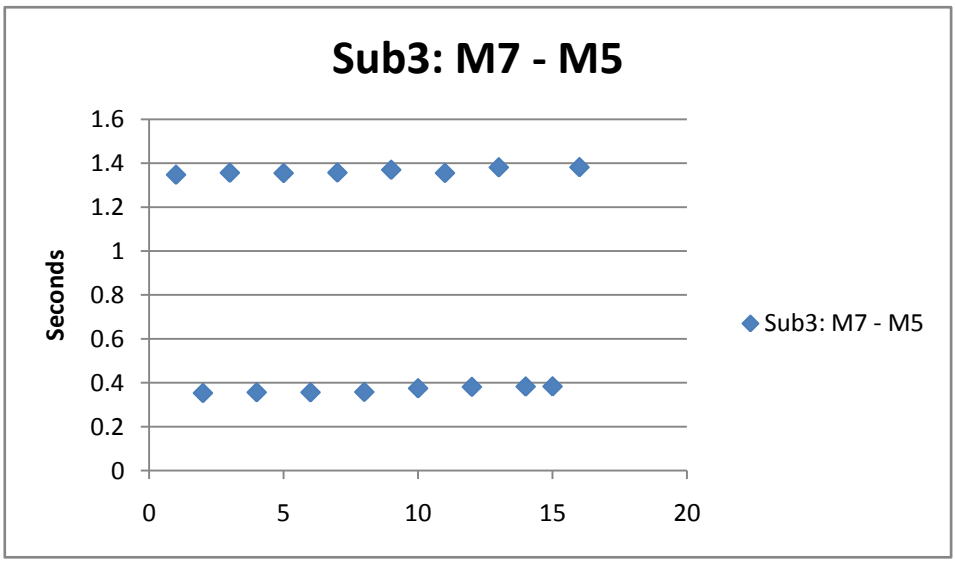

Figure 19. The relative delay (in seconds) between M7-M5 for Sub3

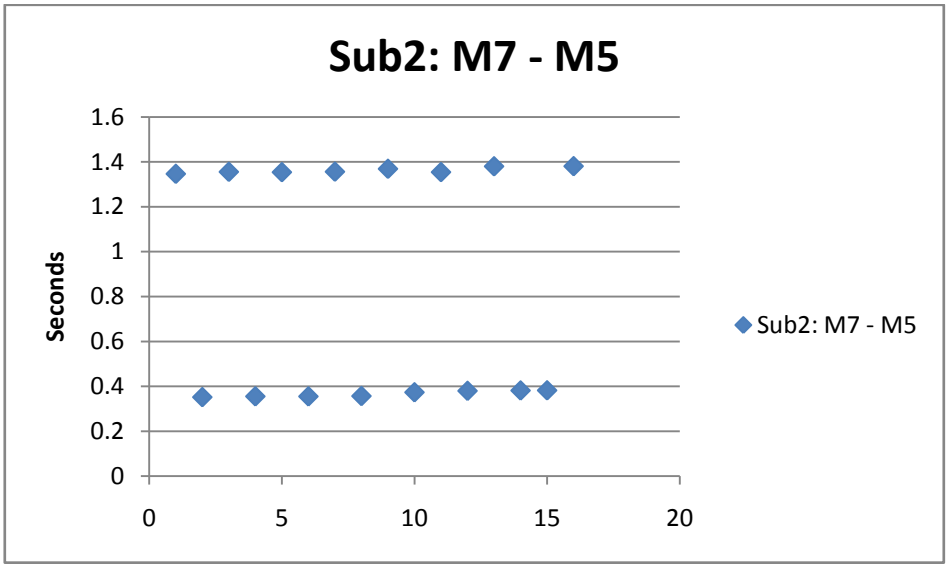

Figure 20. The relative delay (in seconds) between M7-M5 for Sub2

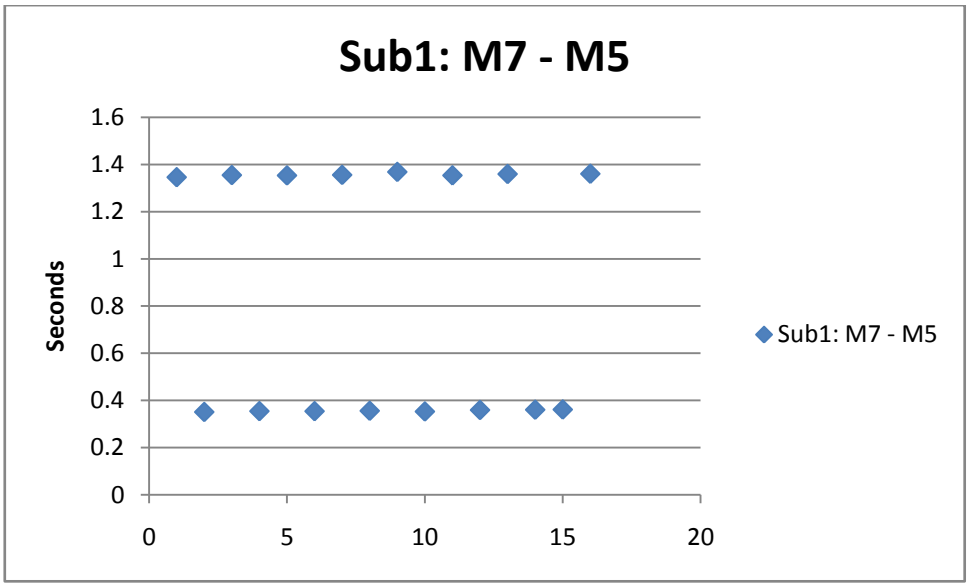

Figure 21. The relative delay (in seconds) between M7-M5 for Sub2

6.4 Multiple Watchers for multiple events

In this scenario as in the previous one, there are three Watchers (Sub1, Sub2, and Sub3), but two of them (Sub1 and Sub2) subscribed to (i.e., are interested in) the 'Location' event and Sub3 subscribed to the 'Presence' event. Two different applications running on behalf of the PUAs send different Publish messages for both 'Location' and 'Presence' events, containing new context information. This scenario was used to: (a) measure the server's response time and as in the previous scenarios to see whether it changes or not, and (b) determine whether the server sends the Notify messages correctly (i.e., only to interested

Watchers). For example if the Publish messages concerns the 'Location' event, then the server should send a Notify message related to this Publish message only to Watchers who subscribed to the 'Location' event. In this case we send Publish messages with two different event fields in a short time interval (the new Publish message is sent before the previous one expires) to see if the server can handle different events at the same time or not. The messages flow is similar to the previous scenario (figure 17), except that here there are two PUAs. Table 11 shows the messages received/send by the server (note that the ✓ mark in the 'Event' column shows that the server correctly sends the Notify to only the interested Watchers for this event).

Table 11. Wireshark output for multiple Watchers and multiple events

No.	Time	Source	Destination	Protocol	Info	Event
M1	2.652763	130.237.239.112	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238	Location
M2	2.669525	130.237.15.238	130.237.239.112	SIP	Status: 200 OK	
M3	3.121144	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	
M4	3.189704	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M1	4.789747	130.237.238.96	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238	Location
M2	4.790801	130.237.15.238	130.237.238.96	SIP	Status: 200 OK	
M3	5.249014	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	
M4	5.31714	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M1	7.007671	130.237.238.242	130.237.15.238	SIP	Request: SUBSCRIBE sip:ccsleft@130.237.15.238	Presence
M2	7.014092	130.237.15.238	130.237.238.242	SIP	Status: 200 OK	
M3	7.497009	130.237.15.238	130.237.238.242	SIP/XML	Request: NOTIFY sip:Sub3@130.237.238.242	
M4	7.593033	130.237.238.242	130.237.15.238	SIP	Status: 200 OK	
M5	26.10331	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Location
M6	26.10634	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	27.4336	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	✓
M7	27.45465	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	✓
M8	27.49844	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M8	27.52589	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M5	31.10034	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Location
M6	31.10785	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	31.43342	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	✓
M7	31.43428	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	✓
M8	31.49806	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M8	31.50627	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	

M5	36.10073	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Location
M6	36.10483	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	37.43354	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	✓
M7	37.45467	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	✓
M8	37.49871	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M8	37.52698	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M5	283.6687	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Location
M6	283.6749	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	285.4348	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	✓
M7	285.4555	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	✓
M8	285.4979	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M8	285.5195	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M5	288.7724	130.237.15.196	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Presence
M6	288.7777	130.237.15.238	130.237.15.196	SIP	Status: 200 OK	
M7	289.4344	130.237.15.238	130.237.238.242	SIP/XML	Request: NOTIFY sip:Sub3@130.237.238.242	✓
M8	289.5022	130.237.238.242	130.237.15.238	SIP	Status: 200 OK	
M5	293.667	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Location
M6	293.6704	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	295.4386	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	✓
M7	295.4593	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	✓
M8	295.5059	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M8	295.5263	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M5	303.661	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Location
M6	303.6669	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	305.4343	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	✓
M7	305.4353	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	✓
M8	305.4974	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M8	305.5215	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	
M5	308.5678	130.237.15.196	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Presence
M6	308.5694	130.237.15.238	130.237.15.196	SIP	Status: 200 OK	
M7	309.4339	130.237.15.238	130.237.238.242	SIP/XML	Request: NOTIFY sip:Sub3@130.237.238.242	✓

M8	309.5015	130.237.238.242	130.237.15.238	SIP	Status: 200 OK	
M5	313.6713	130.237.239.73	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238	Location
M6	313.6738	130.237.15.238	130.237.239.73	SIP	Status: 200 OK	
M7	315.4346	130.237.15.238	130.237.238.96	SIP/XML	Request: NOTIFY sip:Sub2@130.237.238.96	✓
M7	315.4356	130.237.15.238	130.237.239.112	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.112	✓
M8	315.5082	130.237.239.112	130.237.15.238	SIP	Status: 200 OK	
M8	315.5087	130.237.238.96	130.237.15.238	SIP	Status: 200 OK	

If you look at the 'info' and 'Event' column of table 11, you will see that whenever the server receives a Publish message with 'Location' event, it only notifies Sub1 and Sub2 (those who subscribed for a 'Location' event) and the server does not send the Notify message to Sub3. On the other hand, when there is an update for the 'Presence' event, only Sub3 is notified. Therefore the results indicate that as a presence server, our server can handle multiple Watchers (for different events) along with multiple Publish messages correctly. Figure 22 compares the server response time (the time from receiving the Publish messages until sending the Notify message), by showing the relative delay between these messages (i.e., M7-M5) for Watchers Sub2 and Sub1.

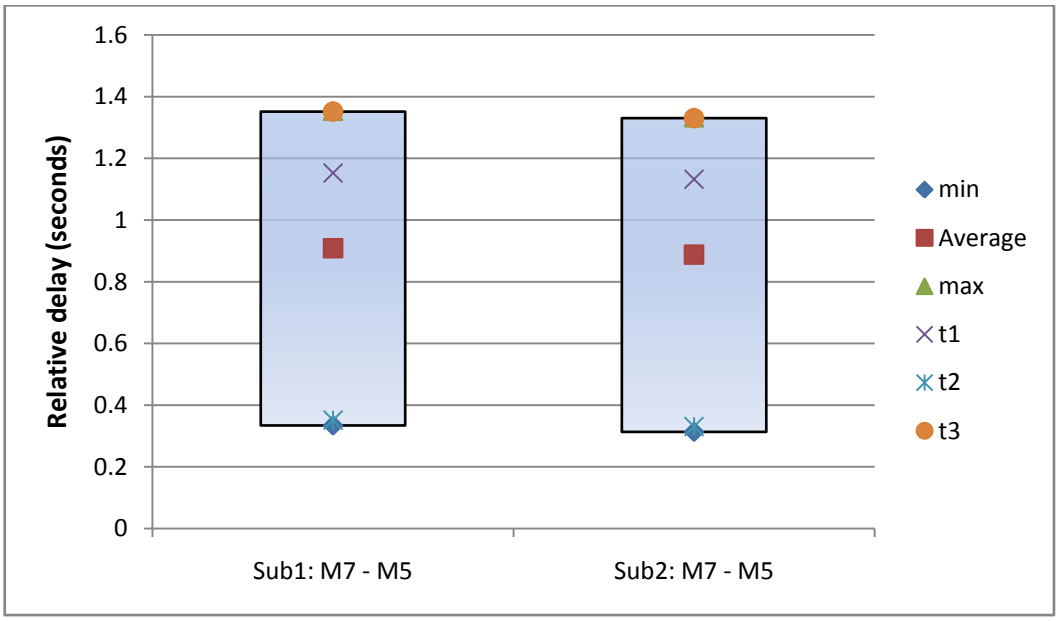

Figure 22. The relative delay (in seconds) between M7-M5 for Sub2 and Sub1; results are shown for the three sets of measurements t1 .. t3.

Comparing figures 22 and 18 indicates that the server's response time does not significantly change when there are different Watchers for different events. Therefore subscribing of the Watchers to different events (and receiving Publish messages for different events by the server) does not increase the server's response time.

6.5 Flooding the server with the Subscribe messages

In the final test scenario I first tried to flood the server with Subscribe messages to see (a) if the server can handle a large number of Watchers, and (b) to examine how the server's response changes. To generate large number of subscriptions, I used four PDAs subscribing to the server. The messages were captured with Wireshark. This captured file was used with the 'netdude' software [66] to edit the saved subscription packets. Then by using the 'tcpreplay' software [67] I was able send 1,000,000 Subscribe message (in less than 10 minutes) to the server in a short period of time in order to simulate a large number of Watchers. I was not able to determine whether the server properly handled all of the Subscribe messages (because of the limitation of the PUAs to reply with OK messages to the server), but looking at the server in debugging mode indicated that the server did not crash even with this huge number of Watchers; as it seemed to correctly parse and handle each of these messages. Note that this rate is roughly 1667 subscriptions per second.

In the second stage I used three PDAs (Sub1, Sub2, and Sub4) and tried to subscribe through each one, 20 times (emulating 60 Watchers). The server accepted multiple subscription messages from the same PDA, if the PDA sends the requests at different times, and server added a new record for this subscription in its database (just as it would do for any new Watcher. This may cause problem, but at the same time it could be a feature, anyway this can be addressed by policy rules to at least limited subscription accepted from one Watcher to prevent subscription flood from one user). After the server received these 60 Watchers' Subscribe messages I started Wireshark and sent a Publish message with a 'Location' event to see how the server processed this Publish message and to see if all 60 Watchers received their notification. Table 12 shows the Wireshark output for this test.

Table 12. Wireshark output showing the server's response following a PUBLISH, when there were 60 Watchers.

No.	Time	Source	Destination	Protocol	Info
PUBLISH	*REF*	130.237.15.196	130.237.15.238	SIP/XML	Request: PUBLISH sip:ccsleft@130.237.15.238
OK	0.005276	130.237.15.238	130.237.15.196	SIP	Status: 200 OK
Watcher1	1.229377	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher2	1.230276	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher3	1.231066	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher4	1.231838	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher5	1.232721	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher6	1.234135	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher7	1.234966	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY

					sip:Sub4@130.237.239.32
Watcher8	1.235747	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher9	1.236618	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher10	1.237438	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher11	1.242114	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher12	1.243045	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher13	1.243985	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher14	1.244912	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher15	1.245925	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher16	1.24686	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher17	1.247756	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher18	1.248662	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher19	1.249622	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher20	1.250587	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher21	1.251487	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher22	1.252395	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher23	1.258285	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher24	1.259265	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher25	1.260195	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher26	1.261142	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher27	1.262149	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher28	1.263089	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher29	1.264004	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher30	1.264993	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher31	1.265882	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY

					sip:Sub4@130.237.239.32
Watcher32	1.266775	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher33	1.267689	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher34	1.268707	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher35	1.269671	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher36	1.270574	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher37	1.271533	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher38	1.272426	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher39	1.273408	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher40	1.274322	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher41	1.275301	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher42	1.276218	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher43	1.277344	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher44	1.27824	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher45	1.279191	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher46	1.280116	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher47	1.281092	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher48	1.282056	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher49	1.282944	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher50	1.283833	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher51	1.284783	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher52	1.287282	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher53	1.299827	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher54	1.300758	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher55	1.301721	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY

					sip:Sub4@130.237.239.32
Watcher56	1.30262	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher57	1.303502	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175
Watcher58	1.304388	130.237.15.238	130.237.239.32	SIP/XML	Request: NOTIFY sip:Sub4@130.237.239.32
Watcher59	1.305439	130.237.15.238	130.237.239.219	SIP/XML	Request: NOTIFY sip:Sub1@130.237.239.219
Watcher60	1.306354	130.237.15.238	130.237.239.175	SIP/XML	Request: NOTIFY sip:Sub2@130.237.239.175

Table 12 shows two valuable results. (a) It shows that the server correctly handled all of the Watchers as it properly sent the Notify messages to all of them. (b) As mentioned in the third scenario each Watcher received the Notify message with an additional 1 ms delay. In this scenario the last Watcher (Watcher60) receives the Notify message 77 ms after the first Watcher receives its Notify message. Most Watchers received a Notify message after 1 ms of delay. However, there are three exceptions out of these 60 samples (see Figure 23 and 24), which increase the average delay between Notify messages increases to 1.28ms from the 1 ms of all the other interval delays of NOTIFY messages. (Note that this 1ms delay is inherent in the performance of this server and it is not parameterized in the code of the server)

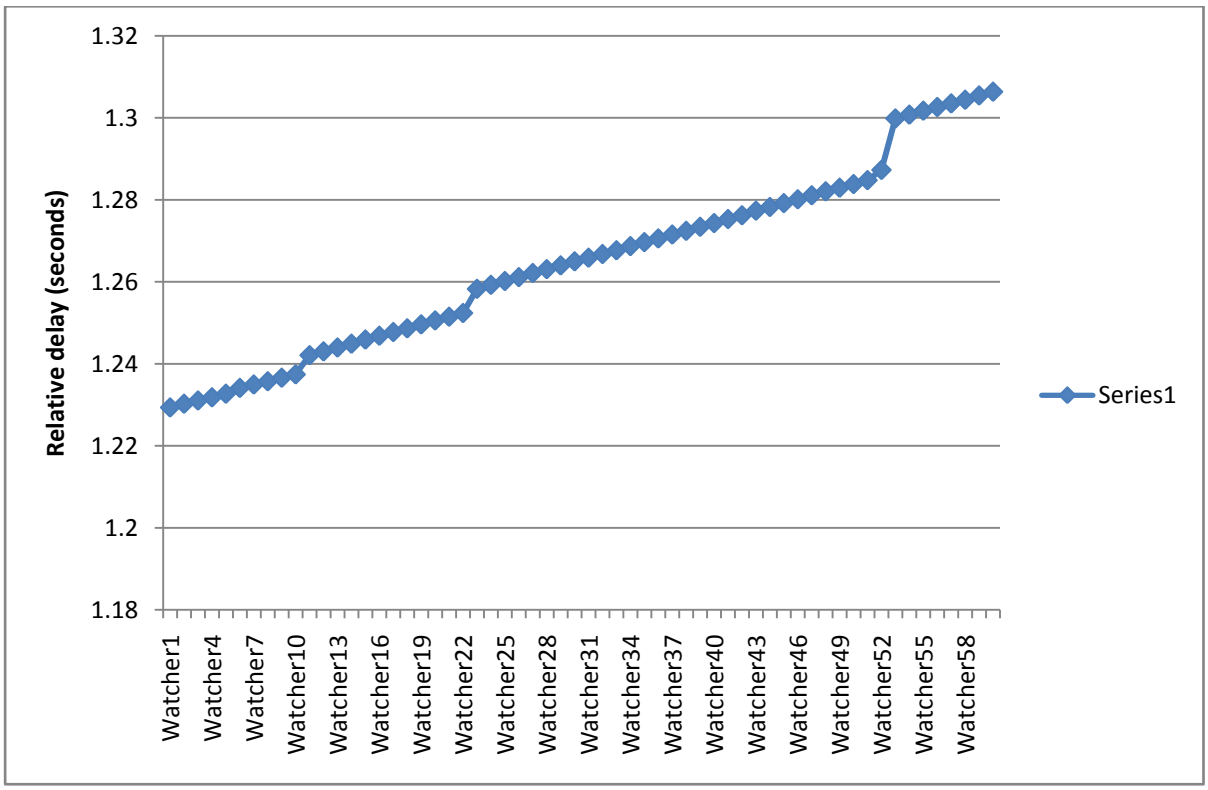

Figure 23. The delay in receiving Notify messages for each of the Watchers

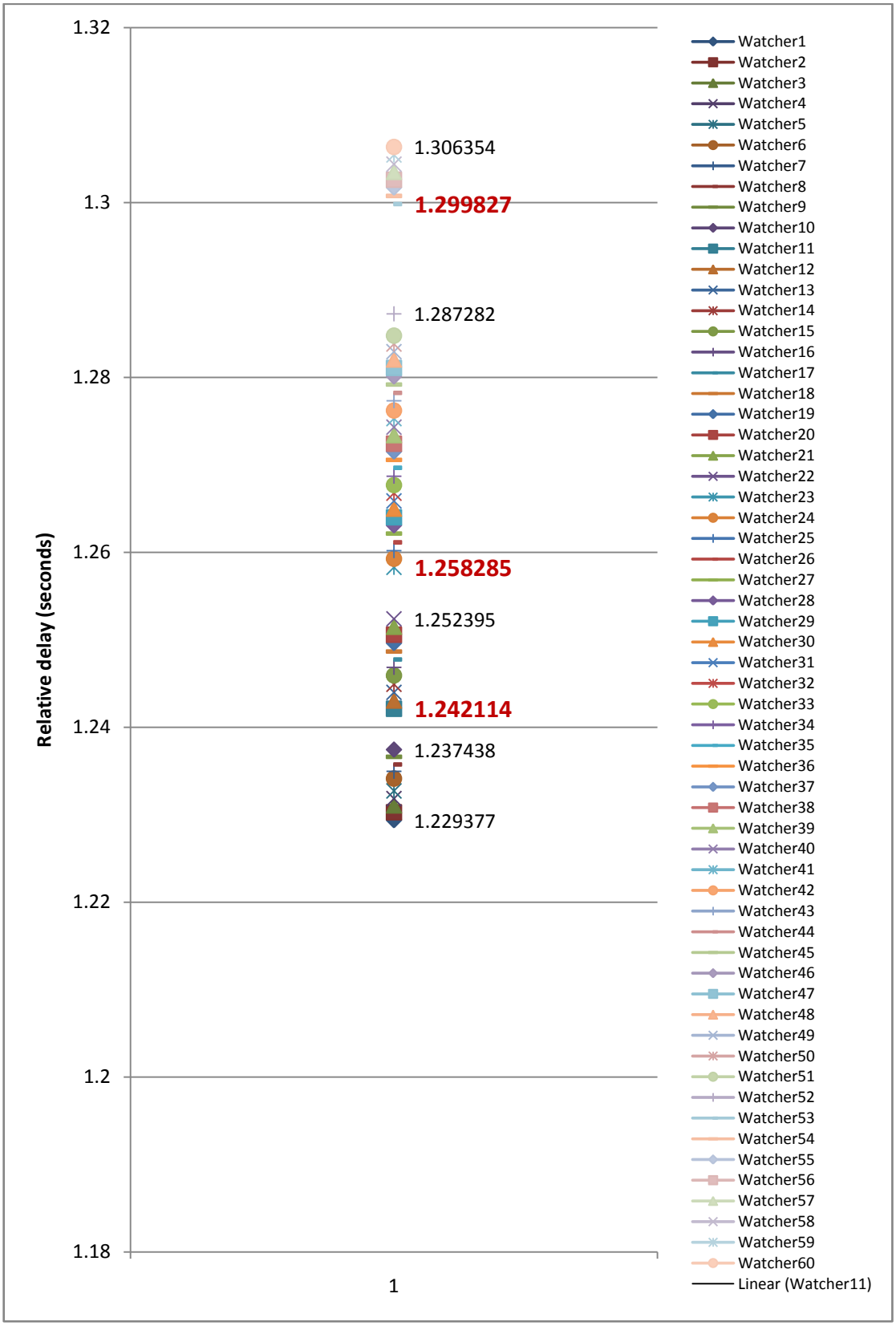

Figure 24. The received Notify messages time for all of the Watchers. The three exceptions are shown by the time of occurrence

6.6 Maximum rate at which the server can send messages

In this section a simple program has been written which just sends 1,000,000 UDP packets from the server (the server host) to a specific destination. These messages are captured with Wireshark in order to calculate the rate at which the server can send these messages. This represents the fastest that the server can response (as it is doing no calculations, database access, etc.). Then we can compare our 1ms value from the previous scenario with this minimal inter packet interval.

Results show that (Figure 25) the server can send in average ~54984 packets per second (which means there is ~18-20 microseconds delay between each of these messages), which is very small compare with 1ms. In addition, if you look at the Figure 26 where I plot 60,000 numbers of packets from these 1,000,000 packets, you can see there are two exceptions. So it seems that, the three exceptions in the previous scenario come from server behavior other than that simply associated with sending of packets.

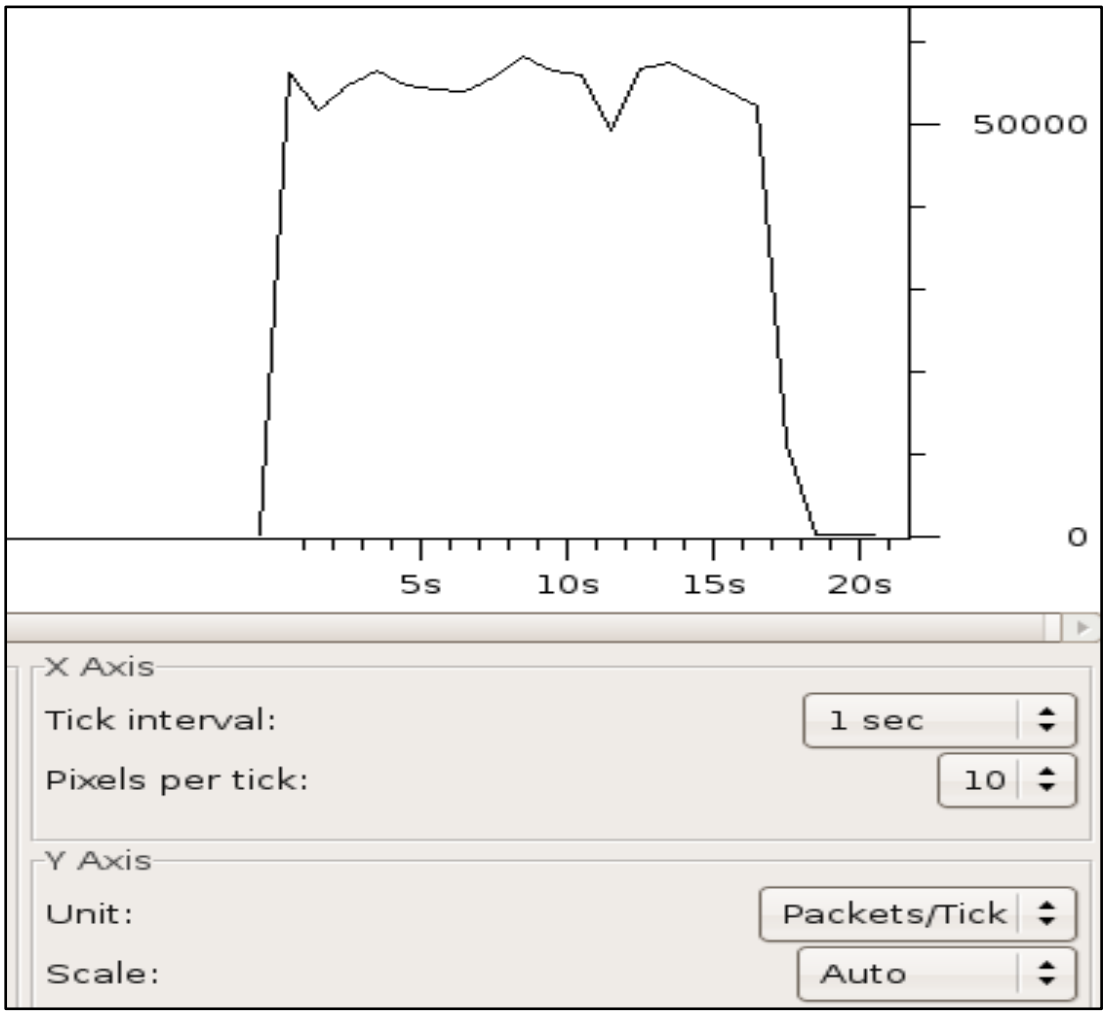

Figure 25. The average number of packets in 1 second while sending 1000000 simple UDP packets from the server to a specific destination.

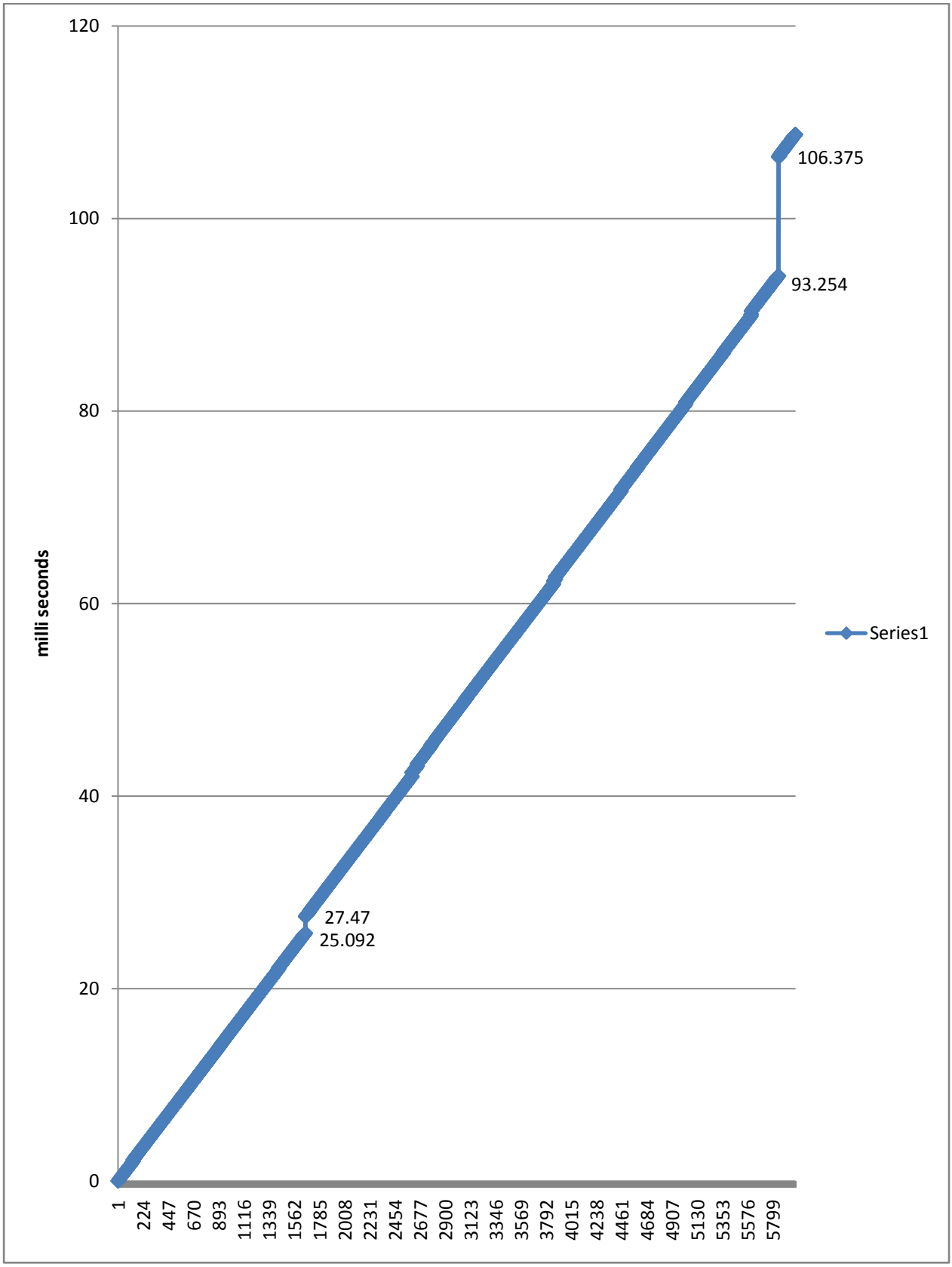

Figure 26. The relative delay for 60,000 packets (as a sample from 1,000,000 UDP packets) from the server to a specific destination. Two exceptions are shown by the time of occurrence

6.7 Some considerations due to MySQL crashes

Before summarizing the evaluation, there is an important issue concerning the SER and its relation with the MySQL, which should be considered. There are a lot of queries from SER to MySQL in different applications. However if we try to read/write from/to wrong place of the database, SER will immediately crash without reporting the problem. Although you can use the "LOG(L_ERR, "string") command (see section 5.2.2) to find out exactly where the problem occurred, when the server crashes it prints out a lot of information, but it is difficult to find our debugging output among this large amount of information (note that you can process the debugging output in an editor where you can search for your debugging strings. Therefore make your debugging output sufficiently unique so that you can easily find them. Moreover, if the server crashes, the old records (from earlier Publish or Subscribe messages before the server crashed) must be removed manually from the database before starting the server. Because these old records are sometimes not automatically eliminated when the server starts again and they may affect the server's functionality (note that you should not recreate the DB after any crash because it will remove the information such as domain name in "domain" and user name in "domain_attrs" tables of SER database which is added in section 5.2.1. So you only need to write a script to delete some of the tables' data which is used in your application such as 'Watcherinfo', 'presentity', etc.)

6.8 Summary of all of the evaluations

By analyzing the results of sections 6.1 and 6.2, we learned that one unit in the field of 'Expire' value from the Subscribe and Publish messages means approximately 1 second at the server. Also we realized that if the Watcher sends the Subscribe message, and after receiving the OK and Notify message from the server it does not reply with an OK message, then the server sends Notify messages continuously. When there is a limited number of Watchers, this behavior of the server is not a big problem (because the server only will send the Notify messages until Subscribe messages has expired), but if we have a lot of Watchers in the system which do not reply with an OK message, then it can be a real bottleneck for the server.

As a point of scalability, we saw that the server can accept and handle the Subscribe and Publish messages within less than 5ms. But there is a rate limit for Notify message about one per second (it is configurable in the server's ser.cfg file). Therefore, even though the server can receive Publish messages for a specific event so quickly, it cannot notify Watchers for the updates of this event within less than one second. However, the server notifies Watchers about the latest context information, which is desired. The server can handle a large number of Watchers (we tested with simulating 1,000,000 Watchers) without crashing. Also it seems that if there is a lot of Watchers in the server database (subscribed for the specific event) and the server receives a new Publish message, the server will Notify all of the Watchers correctly and each Watcher will receive the Notify message with about 1ms delay.

The evaluation reveals that there is a bottleneck, when the number of Publish messages is increasing. So if the server receives different Publish messages so frequently (and if these Publish messages have a high value for 'Expire' value, so the server has to keep them in the database), the server response time will increase when it receives the new Publish message.

7 Conclusions and Future work

7.1 Conclusion

Context information enables humans to create more intelligent devices. These smart devices acquire context information around us and after processing it, they can decide/work automatically on behalf of us. This master thesis enabled me to explore my interest in 'Context awareness' and caused me to study this topic in some depth. Initially, I tried to create my own context aware server building upon some **open source** code. After this, I decided to use the combination of a SER server and MySQL database (for storing context states and subscriptions). First I tried to create my own context module (see section 5.1), but then I learned that a 'presence agent' (PA) module (see section 5.2) had been released by ipte.org with almost the desired functionalities. However, as this PA module supported only one event package (presence), I modified it (see section 5.2.2) in order to support different type of events (e.g. location). As a result of this process I learned that it is much better to build upon the work of others, making modifications as necessary, rather than trying to build a complex system from scratch. However, the effort of trying to build both a complete code base and the presence module gave me insight into how such software needed to respond to the SIP messages.

I used SIP-SIMPLE protocol (an extension of SIP to support instant messaging and presence) to distribute context information among entities. SIP-SIMPLE (see section 2.4) supports presence functions and allows users to be notified of changes in user presence by subscribing to events. For this purpose, three different types of SIP messages were used: (a) the Publish message: for publishing changes in the status of event from PUA to the server, (b) the Subscribe message: for subscribing the Watchers to the server, and (c) the Notify message: for Notifying Watchers by the server, when the server receives a Publish message about updated context information. Along with using SIP-SIMPLE for sending/receiving context information, PIDF (see section 3.4.31) is used as a context model to transfer this context information (presence documents) in a standard format. Although PIDF has its own basic tags which can be used for this purpose (e.g., the <note> tag is used for transferring context information), it has been extended in this thesis to support special tags for our purposes (e.g., for the 'Location' event we added our own specific tags such as: <room>, <floor>, etc.). The SIP-SIMPLE mechanism proved to be well suited to its use in this project. Additionally, PIDF proved to be very easy to extend to support our location event information.

Generally our server: (a) accepts and registers Watchers for different events, using Subscribe messages, sent by different applications to the server, (b) obtains the updated context information for different types of events from Presence User Agents (PUAs) via Publish messages, (c) reads, processes, and stores this context information in the local MySQL database SER, (d) notifies only interested Watchers about this context information via Notify messages, and (e) removes expired Watchers and context information from its

database and Notifies the Watchers about this expiration. Thus in all cases the server showed the correct functional behavior.

In the evaluation phase, I tested: (a) the accuracy of the server (i.e., does the server sends the correct Notify messages only to interested Watchers?), (b) the performance of the server in terms of responsiveness time (i.e., how long does it take for the server to response to each of the different messages), and (c) the scalability of the server (i.e., multiple Watchers try to access the same context). The measurements show that increasing the number of Publish messages for different events in the SER data base, increases the server's response time. Also evaluation shows that, reading/writing from/to the MySQL must be used carefully, otherwise it causes to server crashes. Along with my thesis, there were two parallel theses (Yu Sun [64] and Hubinette Daniel [63]) what are creating context-aware applications. Their applications utilize my server as a context aware server; their efforts show the value of the proposed server and serve as a real demonstration of its operational state and usefulness.

This thesis project forced me to improve my programming ability. In addition, I learned about some new technologies, particularly SIP-SIMPLE, SER and its modules, and PIDF. However, there remain a number of open questions and issues which should be addressed. In the next section I highlight some of the possible future work for someone who would like to continue this work.

7.2 Future work

There are many interesting opportunities for future work building upon this thesis project. Some of the most important future improvements are:

- **Authentication of Watchers:** A next step would be to implement some kind of security mechanism in order to authenticate Watchers, after receiving subscribe messages at the server, before registering the Watchers in the server's database.

- **Authorization of Watchers:** Adding policy based processing of requests to the server is potential area for improvement. In order to authorize different Watchers for different context information, some policy mechanism should be added, so that a policy could be consulted to decide if a given Watcher should be sent a Notify message. For example if we do not want to reveal the location of teachers to students (even students subscribed to the teacher's 'Location' event), then a policy mechanisms should be added to the server, and these policy should be checked by the server before notifying students about the teacher's updated location information. The Geopriv [68] [69] may be relevant to this effort.

- **Policies:** Policies also could be used for security purposes. For example if a Watcher subscribes and unsubscribes to the server continually in a short time, the server could add this Watcher to a black list (or ignore its Subscribe messages for a specific time, due to fact that this Subscribe/unsubscribe probably occurs because of problems at a Watcher). Such a policy could be used to reduce the probability of a denial of service attack.

- **SIP Request/Reply:** In this master thesis I used SIP-SIMPLE for context distribution. There are some other alternatives, such as SIP Request/Reply. In this method unlike SIP-SIMPLE, Watchers receive the context information whenever they request it (rather than when there is a change in the status of events). One should implement such a Request/Reply method for context distribution and compare the results (the performance) with the current implementation (using SIP-SIMPLE). However, it seems both of the methods have some advantages and disadvantages, and based on different applications' requirements, a suitable choice could be made for this specific application.

References

[1] Athanasios Karapantelakis, Adaptive Context-Aware Services for a University Campus, Licentiate thesis proposal (draft), School of Information and Communication Technology (ICT), Royal Institute of Technology (KTH), May 2007.

[2] M. E. Anagnostou, M.A. Lambrou, and E. D. Sykas, Context Aware Service Engineering In Support Of Future Business Networks, Available at: http://context.upc.es/Papers/ContextCOCONET.pdf

[3] Anind K. Dey, Gregory D. Abowd, and Daniel Salber, A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications, Human-Computer Interaction, 2001, Volume 16, pp. 97–166, Available at: http://www.cc.gatech.edu/fce/ctk/pubs/HCIJ16.pdf

[4] Jiri Kuthan. SIP and SER: More than you ever wanted to know About, September 2003, Available at: http://voip.internet2.edu/meetings/slides/200310/SIP_Express_Router.pdf

[5] Harry Lik Chen, An Intelligent Broker Architecture for Pervasive Context-Aware Systems, PhD Thesis, 2004, Department of Computer Science andElectrical Engineering, University of Maryland, Available at: http://ebiquity.umbc.edu/_file_directory_/papers/152.pdf

[6] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg, A survey on context-aware systems, International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC), Vol. 2, No. 4, 2007, page 263-277, Available at: http://www.vitalab.tuwien.ac.at/~florian/papers/ijahuc2007.pdf

[7] G. Q. Maguire Jr. Lecture notes for the course 2G1325, Practical Voice Over IP (VoIP): SIP and related protocols, School of Information and Communication Technology (ICT), Royal Institute of Technology (KTH), Spring 2007, Available at: http://www.imit.kth.se/courses/2G1325/VoIP-2007a.pdf

[8] Wikipedia. RTP protocol, Available at: http://en.wikipedia.org/wiki/Real-time_Transport_Protocol

[9] Harry Chen, Tim Finin, and Anupam Joshia, Context Broker for Building Smart Meeting Rooms, Department of Computer Science & Electrical Engineering, University of Maryland Baltimore County, 2004, Available at: http://ebiquity.umbc.edu/_file_directory_/papers/78.pdf

[10] XML description in Wikipedia, The Extensible Markup Language (XML), Last modified October 2007, Available at: http://en.wikipedia.org/wiki/Xml ,

[11] Quan Z. Sheng, and Boualem Benatallah, ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services, Mobile Business, 2005. ICMB 2005. International Conference on, July 2005, pages (206-212) Available at: http://ieeexplore.ieee.org/iel5/9999/32116/01493610.pdf?tp=&isnumber=&arnumber=1493610

[12] Extensible Markup Language (XML) 1.0 (Fourth Edition): W3C Recommendation, 16 August 2006, edited in place 29 September 2006, Available at: http://www.w3.org/TR/REC-xml/

[13] Henry Sinnreich and Alan B. Johnston. Internet Communications Using SIP Delivering VoIP and Multimedia Services with Session Initiation Protocol, Wiley, Published July 2006, 2nd edition, 408 pages, ISBN 0471776572, pp.142 – 150.

[14] Sergi Laencina Verdaguer. Model driven context awareness, Master thesis, School of Information and Communication Technology (ICT), Royal Institute of Technology (KTH), January 2007.

[15] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. "SIP: Session Initiation Protocol", RFC 2543, March 1999, Available at: http://www.ietf.org/rfc/rfc2543.txt

[16] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. "SIP: Session Initiation Protocol", RFC 3261, June 2002, Available at: http://www.ietf.org/rfc/rfc3261.txt

[17] Iptel.org: SIP Express Router, Available at: http://www.iptel.org/ser/

[18] Thomas Strang and Cluadia Linnhoff-Popien, "A context modeling survey," in First International Workshop on Advanced Context Modelling, Reasoning and Management (UBICOMP), September 2004, Available at: http://www.itee.uq.edu.au/~pace/cw2004/Paper15.pdf

[19] Jonathan Rosenberg, Jonathan Lennox, and Henning Schulzrinne. Programming Internet Telephony Services, IEEE Network Volume 13, Issue 3, May-June 1999, pp. 42 – 49.

[20] Unified Modeling Language (UML), Last updated April 2007, Available at: http://www.omg.org/technology/documents/formal/uml.htm

[21] Oukhay Younes. Context Aware Services, Master thesis, School of Information and Communication Technology (ICT), Royal Institute of Technology (KTH), January 2006.

[22] OpenSER: the Open Source SIP Server, Available at: http://www.openser.org/, Last visited: July 2007

[23] Paul Hazlett, Simon Miles, and Greger V.Teigre. SER-Getting Started, Last revised draft: February 2006, Available at: http://siprouter.onsip.org/doc/SER-GettingStarted.pdf

[24] Wikipedia. SIP Express Router (SER), Last modified: October 2007, Available at: http://en.wikipedia.org/wiki/SIP_Express_Router_%28SER%29,

[25] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo, Reflective Middleware Solutions for Context-Aware Applications, Dept. of Computer Science, University College London, September 2001, Available at: http://www.cs.ucl.ac.uk/staff/c.mascolo/www/ref.pdf

[26] Jason I. Hong and James A. Landay, An Infrastructure Approach to Context-Aware Computing, University of California at Berkeley, Human-Computer Interaction (HCI) Journal 2001. 16(2-4) 2001, available at: http://www.leaonline.com/doi/pdf/10.1207/S15327051HCI16234_11

[27] J. Rosenberg, A Presence Event Package for the Session Initiation Protocol, SIP-SIMPLE, RFC 3856, Aug. 2004, Available at: ftp://ftp.rfc-editor.org/in-notes/rfc3856.txt

[28] Hannu-Pekka Rajaniemi and Kliment Yanev, “SIP and Presence”, Seminar on Instant Messaging and Presence Architectures in the Internet, University of Helsinki, Department of Computer Science, September 2005, Available at: http://www.cs.helsinki.fi/u/yanev/simplep.pdf

[29] H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr, and J. Peterson, Presence Information Data Format (PIDF), RFC 3863 (Proposed Standard) , Aug. 2004, Available at: http://www.ietf.org/rfc/rfc3863.txt

[30] Printer Job Language Technical Reference Manual, Hewlett Packard, Edition 12, June 2003, Available at: http://stuff.mit.edu/afs/sipb/contrib/doc/HP/8150/pjlref.pdf

[31] npadmin homepage, July 30 1999, Available at: http://npadmin.sourceforge.net/#top

[32] B. Schilit & M. Theimer, Disseminating Active Map Information to Mobile Hosts, IEEE Network, September/October 1994.

[33] P.J Brown , J.D. Bovey, and X. Chen, Context-Aware Applications: From the Laboratory to the Marketplace, IEEE Personal Communications, 4(5):58-64, October 1997, Available at: http://www.comsoc.org/pci/private/1997/oct/pdf/Brown.pdf

[34] Bill N. Schilit, Norman Adams, and Roy Want, Context-Aware Computing Applications, Workshop on Mobile Computing Systems and Applications, 1994. Proceedings, Volume, Issue, 8-9 Dec 1994 Page(s):85 - 90, Available at: ftp://ftp.parc.xerox.com/pub/schilit/wmc-94-schilit.ps

[35] Anind K. Dey, Understanding and Using Context, Personal and Ubiquitous Computing Journal, Volume 5 (1), 2001, pp. 4-7 Available at: http://www.cc.gatech.edu/fce/ctk/pubs/PeTe5-1.pdf

[36] Karen Hendricksen, Jadwiga Indulska, and Andry Rakotonirainy, Generating context management infrastructure from high-level context models, Proceedings of the 4th International Conference on Mobile Data Management, January 2003, pages (1–6), Available at: http://coblitz.codeen.org:3125/citeseer.ist.psu.edu/cache/papers/cs/27262/http:zSzzSzwww.dstc.edu.auzSzm3zSzpaperszSzContextManagement.pdf/henricksen03generating.pdf

[37] Object-Role Modeling (ORM), Last updated: July 2006, Available at: http://www.orm.net/

[38] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, and Josef Altmann, Context-awareness on mobile devices - the hydrogen approach, System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on, Jan. 2003, Available at: http://ieeexplore.ieee.org/iel5/8360/26341/01174831.pdf?tp=&isnumber=&arnumber=1174831

[39] Composite Capability/Preference Profiles (CC/PP), W3C Recommendation 15 January 2004, Available at: http://www.w3.org/TR/CCPP-struct-vocab/

[40] J.Lennox, X.Wu, and H.Schulzrinne. CPL: A Language for User Control of Internet Telephony Services, RFC 3880: October 2004: Available at: http://www.ietf.org/rfc/rfc3880.txt

[41] Jose Costa Requena, Helsinki University of Technology, XML for creating 3G Services (CPL), 2000, Available at: http://mia.ece.uic.edu/~papers/WWW/MultimediaStandards/cpl_xml.pdf

[42] CPL XML DTD draft, January 2002, Available at: http://xml.coverpages.org/CPL-DTD-200201.txt,

[43] A. Devlic, "Extending CPL with context ontology", In Mobile Human Computer Interaction (Mobile HCI 2006) Conference Workshop on Innovative Mobile Applications of Context (IMAC), Espoo/Helsinki, Finland, September 2006, Available at: http://web.it.kth.se/~devlic/article.pdf

[44] Alisa Devlic. CPL extensions, Report for course 2G1325/2G5564, Royal Institute of Technology (KTH), 31 May 2006.

[45] Using UAProf (User Agent Profile) to Detect User Agent Types and Device Capabilities, Learn about UAProf by Developers 'Home webpage, Last visited: July 2007, Available at: http://www.developershome.com/wap/detection/detection.asp?page=uaprof

[46] Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation 10 February 2004, Available at: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[47] OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004, Available at: http://www.w3.org/TR/owl-features/

[48] Semantic Web, semantic web home page in W3C, Available at: http://www.w3.org/2001/sw/

[49] IM description in Wikipedia, Instant messaging (IM), Available at: http://en.wikipedia.org/wiki/Instant_messaging

[50] Printer Working Group (PWG) Semantic Model January 2004, Available at: ftp://ftp.pwg.org/pub/pwg/candidates/cs-sm10-20040120-5105.1.pdf

[51] M. Day, J. Rosenberg, and H. Sugano, A Model for Presence and Instant Messaging, RFC 2778, Feb. 2000, Available at: http://www.ietf.org/rfc/rfc2778.txt

[52] H. Schulzrinne, V. Gurbani, P. Kyzivat, and J. Rosenberg, RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF), RFC 4480, July 2006, Available at: http://www.ietf.org/rfc/rfc4480.txt

[53] M. Lonnfors, E. Leppanen, H. Khartabil, and J. Urpalainen, Presence Information Data format (PIDF) Extension for Partial Presence, Internet-Draft, November 2006, Available at: http://www.ietf.org/internet-drafts/draft-ietf-simple-partial-pidf-format-08.txt

[54] H. Schulzrinne, Timed Presence Extensions to the Presence Information Data Format (PIDF) to Indicate Status Information for Past and Future Time Intervals, RFC 4481, July 2006, Available at: http://www.ietf.org/rfc/rfc4481.txt

[55] The Printer Working group (PWG), PWG home page, Available at: http://www.pwg.org/

[56] R. Herriot, R. deBry, S. Isaacson, P. Powell, and T. Hastings (Editor), Internet Printing Protocol/1.1: Model and Semantics, RFC 2911, Sep. 2000, Available at: http://www.ietf.org/rfc/rfc2911.txt , see also RFC 2567, RFC 2568, RFC 2569, and RFC 2910

[57] HP Web Jet admin software - overview and features, Available at: http://h20338.www2.hp.com/hpsub/cache/332262-0-0-225-121.html?jumpid=ex_r2845_go/webjetadmin/gc121306

[58] Extensible Markup Language (XML), XML Tutorial, W3 Schools web page, Available at: http://www.w3schools.com/xml/default.asp

[59] C.-G. Jansson, M. Jonsson, T. Kanter, F. Kilander, G. Maguire, and Li Wei, Adaptive connectivity management middleware for heterogeneous wireless networks, Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st, Volume 5, Issue , 30 May-1 June 2005 Page(s): 2954 - 2958 Vol. 5, Digital Object Identifier 10.1109/VETECS.2005.1543888, Available at: http://ieeexplore.ieee.org/iel5/10360/32962/01543888.pdf?arnumber=1543888

[60] Andreas Wennlund, "Context-aware wearable device for reconfigurable application networks", M.Sc. thesis, Royal Institute of Technology (KTH), Stockholm Sweden, March 2003.

[61] Jan Janak, Jiri Kuthan, and Bogdan Iancu, SIP Express Router v0.8.8 - Developer's Guide, 2002, Available at: http://old.iptel.org/ser/doc/serdev/serdev.html#MODULE-INTERFACE

[62] A. Niemi, Session Initiation Protocol (SIP) Extension for Event State Publication, RFC 3903, October 2004, Available at: http://www.ietf.org/rfc/rfc3903.txt

[63] Daniel Hubinette, Occupancy Sensor System: for Context-aware computing, Royal Institute of Technology (KTH), Department of Communication Systems, December, 2007.

[64] Yu Sun, Context-aware applications for a Pocket PC, Royal Institute of Technology (KTH), Department of Communication Systems, December, 2007.

[65] Athanasios Karapantelakis, Alisa Devlic, Mohammad Zarifi Eslami and Saltanat Khamit, Printing in pervasive computing environments, Royal Institute of Technology (KTH), Department of Communication Systems, December, 2007.

[66] Netdude (The Network Dump data Displayer and Editor is a framework for inspection, analysis and manipulation of tcpdump trace files), http://netdude.sourceforge.net/

[67] Aaron Turner, Tcpreplay (a suite of BSD licensed tools to use previously captured traffic to test a variety of network devices), http://tcpreplay.synfin.net/trac/

[68] Geographic Location/Privacy (Geopriv), Available at: http://www.ietf.org/html.charters/geopriv-charter.html, last modified: 2007-11-26

[69] J. Cuellar, J. Morris, D. Mulligan, J. Peterson, and J. Polk, Geopriv Requirements, RFC 3693, February 2004, available at: http://www.ietf.org/rfc/rfc3693.txt

[70] Gerald Combs, Wireshark, Web page: http://www.wireshark.org/ last modified 2007-12-02.

[71] F. Andreasen and B. Foster, Media Gateway Control Protocol (MGCP) Version 1.0, RFC 3435, January 2003, Availabe at: http://www.rfc-editor.org/rfc/rfc3435.txt

[72] Niklas Zennström and Janus Friis, Skype, 2003, available at: http://about.skype.com/

Appendix A

Installing and configuring SER and MySQL on Ubuntu.

A.1. Change Ubuntu root password

sudo passwd

and become root:

su root

A.2. MySQL

Make sure you have the required packages in order to compile:

sudo apt-get install g++ libncurses5-dev

A.3.Download the MySQL source from

http://dev.mysql.com/get/Downloads/MySQL-5.0/mysql-5.0.41.tar.gz/from/pick (pick the Swedish mirror)

From the terminal unzip and Install MySQL:

tar zxvf mysql-5.0.41.tar.gz

Go to the folder where mysql is, and issue:

./configure --prefix=/usr/local/mysql

make

make install

then do the following:

cp support-files/my-medium.cnf /etc/my.cnf

cd /usr/local/mysql

chown -R mysql .

chgrp -R mysql .

bin/mysql_install_db --user=mysql

chown -R root .

chown -R mysql var

bin/mysqld_safe --user=mysql &

add the binaries path to your .bashrc file (and the SIP_DOMAIN environment variable, you'll need it later):

export PATH=$PATH:/usr/local/mysql/bin

export SIP_DOMAIN=130.237.15.237

(change 130.237.15.237 is your IP address)

Load the variables to your terminal:

source ~/.bashrc

In order to start mysql issue:

/usr/local/mysql/bin/mysqld_safe &

Change root password for MySQL:

/usr/local/mysql/bin/mysqladmin -u root password 'new-password'

A.4. Compiling SER

Get ser: http://ftp.iptel.org/pub/ser/0.9.6/src/ser-0.9.6_src.tar.gz

Get required packages:

sudo apt-get install bison flex libxml2-dev

Edit the Makefile in the ser directory

Replace the lines:

```
exclude_modules?=               cpl ext extcmd \
                                postgres snmp \
                                im \
```

```
                                jabber mysql \
                                cpl-c \
    auth_radius group_radius uri_radius
```

with:

```
exclude_modules?=               cpl ext extcmd \
                                postgres snmp \
                                im \
                                        jabber \
                                auth_radius group_radius uri_radius
```

then do:

```
make all
make install
```

see if the optional modules are installed properly:

```
ls /usr/local/lib/ser/modules/ |grep ^[mc][yp]
```

configure SER to support authentication:

edit /usr/local/etc/ser/ser.cfg

uncomment the following:

change

```
#loadmodule "/usr/local/lib/ser/modules/mysql.so"
```

to

```
loadmodule "/usr/local/lib/ser/modules/mysql.so"
loadmodule "/usr/local/lib/ser/modules/cpl-c.so"
```

change
#loadmodule "/usr/local/lib/ser/modules/auth.so"
#loadmodule "/usr/local/lib/ser/modules/auth_db.so"

to
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"

change
#modparam("auth_db", "calculate_ha1", yes)

to
modparam("auth_db", "calculate_ha1", yes)

change
#modparam("auth_db", "password_column", "password")

to
modparam("auth_db", "password_column", "password")

change

```
#                   if (!www_authorize("iptel.org", "subscriber")) {
#                           www_challenge("iptel.org", "0");
#                           break;
#                   };
```

to

```
                    if (!www_authorize(130.237.15.237, "subscriber")) {
                            www_challenge(130.237.15.237, "0");
                            break;
                    };
```

replace 130.237.15.237 with your own IP address

Note: You may wish to set write-back to mysql to minimize runtime delay with this parameter:

modparam("usrloc", "db_mode", 2)

Create the SER database:

/usr/local/sbin/ser_mysql.sh create

Start SER:

/usr/local/sbin/ser -E

Add users:

/usr/local/sbin/serctl add <username> <password> <email>

Appendix B

UDP Socket programs (Client and Server) which we will need when we want send data between service agents and SER.

B.1. UDP Client

```
/* datagram client */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* This function reports the error and exits back to the shell */
static void
bail (const char *on_what)
{
       fputs (strerror (errno), stderr);
       fputs (": ", stderr);
       fputs (on_what, stderr);
       fputc ('\n', stderr);
       exit (1);
}

int main (int argc, char **argv)
{
       int z, s /* s:Socketcket */ ;
       char *srvr_addr = NULL;
       struct sockaddr_in adr_srvr /* AF_INET */ ;
       int len_inet;           /* length */
       FILE *fp;
     if ( argc >= 2 ) {
          /* Addr on cmdline: */
          srvr_addr = argv[1];
    }
    else {

          /* Use default address: */
          srvr_addr = "127.0.0.1";
              printf("using loopback !\n");
```

```
    }

/* Create a socket address, to use to contact the server with*/
    bzero(&adr_srvr,sizeof adr_srvr);
    adr_srvr.sin_family = AF_INET;
    adr_srvr.sin_port = htons (9090);
    inet_pton(AF_INET, srvr_addr, &adr_srvr.sin_addr);
    if (adr_srvr.sin_addr.s_addr == INADDR_NONE)
        bail ("bad address.");
    len_inet = sizeof adr_srvr;

/* Create a UDP socket to use*/
    s = socket (AF_INET, SOCK_DGRAM, 0);
    if (s == -1)
    bail ("socket()");
    char * sendline = "Hello";
    sendto(s, sendline, strlen(sendline),0,(struct sockaddr *) &adr_srv     len_inet);
    close(s);
}
```

B.2. UDP Server

```
/* datagram Server */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

/* This function reports the error and exits back to the shell */
static void
bail (const char *on_what)
{
    fputs (strerror (errno), stderr);
    fputs (": ", stderr);
    fputs (on_what, stderr);
    fputc ('\n', stderr);
    exit (1);
}

int main (int argc, char **argv)
{
    int z, s; /* s: Socket */
```

```
        socklen_t len_inet /* length */;
        char mesg[512]; /* Recv buffer */;
        struct sockaddr_in adr_inet /* AF_INET */ , adr_clnt /* AF_INET */;

        /* Create a UDP socket to use */
        s = socket (AF_INET, SOCK_DGRAM, 0);
        if (s == -1)
        bail ("socket()");

        /* Create a socket address, for use with bind(2) */
        bzero(&adr_inet,sizeof(adr_inet));
        adr_inet.sin_family = AF_INET;
        adr_inet.sin_port = htons (9090);
        adr_inet.sin_addr.s_addr = htonl(INADDR_ANY);
        if (adr_inet.sin_addr.s_addr == INADDR_NONE)
        bail ("bad address.");
        len_inet = sizeof adr_inet;

/* Bind a address to socket, so client programs can contact this server */
        z = bind (s, (struct sockaddr *) &adr_inet, len_inet);
        if (z == -1)
        bail ("bind()");

/* Now wait for requests */
        for (;;)
        {
/* Block until the program receives a datagram at our address & port */
                len_inet = sizeof adr_clnt;
                z = recvfrom (s,mesg,   /* Receiving buffer */ sizeof mesg,0,
(struct sockaddr *) &adr_clnt,     /* Addr */ &len_inet);  /* Addr len, in & out */
                printf("receive :%s\n", mesg);
                if (z < 0)
                bail ("recvfrom(2)");

                /* Send the formatted result back to the client program */
z = sendto (s,mesg,       /* The datagram result to snd */ z,      /* data gram lngth */ 0,(struct
sockaddr *) &adr_clnt,len_inet); /*        Client address length */
                if (z < 0)
                bail ("sendto(2)");
        }
/* Close the socket and exit: */
close (s);
}
```

Appendix C

Getting printer information through PJL and MIB queries.

To see how these two methods work, in this section we will demonstrate some examples with their descriptions. You will see A when we use PJL scripts and B with npadmin commands. and we use I for input commands and O for output of each which in our case the output is the response of ”HP laser jet 4000” which is located in COS lab with a IP address of 130.237.15.229.

C.1. Using PJL

- *Printer status*

AI: using ”INFO STATUS” command:

”\x1B%-12345X@PJL INFO STATUS\r\n\x1B%-12345X\r\n”

AO: @PJL INFO STATUS

CODE=35078

DISPLAY= ”POWERSAVE ON”

ONLINE= TRUE

Description:

This command returns online/off line status of printer, the message currently displayed on the control panel, and a status code. By looking at the status codes we can get plenty of useful information (Appendix D in Printer Job Language Technical Reference Manual), for example: 1001 means ”Ready(online)”, 10006 means ”Toner low”, 10007 means ”Canceling Job”, 10023 means ”PROCESSING JOB FROM TRAY X (X = tray code)”, and etc. But after doing some measurements it seems when printer is in ready (idle) state and we send a command to get it, printer start processing our command and it returns ”Processing job” status instead of ”idle”, so we may use npadmin for getting status which seems return logical results in this case.

- *Printer Configuration*

AI: using ”INFO CONFIG” command:

"\x1B%-12345X@PJL INFO CONFIG\r\n\x1B%-12345X\r\n"

AO: @PJL INFO CONFIG
IN TRAYS [2 ENUMERATED]
INTRAY1
INTRAY2
OUTPUT BINS [1 ENUMERATED]
UPPER
PAPERS [12 ENUMERATED]
LETTER
LEGAL
A4
EXECUTIVE
COM10
MONARCH
C5
DL
JISB5
B5
CUSTOM
A5
LANGUAGES [3 ENUMERATED]
PCL
PCLXL
POSTSCRIPT
USTATUS [4 ENUMERATED]
DEVICE
JOB
PAGE
TIMED
MEMORY MANAGEMENT [4 TABLE]
MEMORY AMOUNT
UTILIZATION
SYSTEM 63897600

0

PCL 0 0

POSTSCRIPT 0 0

MEMORY=71303168

DISPLAY LINES=2

DISPLAY CHARACTER SIZE=16

Description:

As we can see, with using just config command we can get plenty of information such as number of tray, supported papers, memory, etc. but we can get theses information individually by just sending specific command (for example if we want to know just memory capacity of printer we can send AI="\x1B%-12345X@PJL INFO MEMORY \r\n\x1B%-12345X\r\n")

C.2. Using npadmin

- Printer status

BI: using ”--status” command:

”npadmin --status 130.237.15.229”

BO: status="idle";

Printer Configuration

BI: To have all information we got by PJL, with npadmin we should use different attributes:

”npadmin --memory --model --name --storage --maxpapersize –input tray --colors --covers 130.237.15.229”

BO:

hostname="130.237.15.229";

model="LaserJet 4000 Series";

memsize="69632";desc="Random AccessMemory";

allocunits="1";size="71303168";used="5337904";allocfail="0";

processColorants="1";

//(1 means it is black/white printer)

maxMediaUnit="micrometers"; maxMediaFeedDir="355600";

maxMediaXFeedDir="215900";

type="sheetFeedAutoRemovableTray";dimUnit="micrometers";

dimFeedDir="297000";dimXFeedDir="210000";capUnit="sheets";

maxCap="500";

curLevel="-3"; //(-3 means that there is enough paper in that try to print at least one more page.)

mediaName="Plain";name="TRAY 2";description="Tray 2";
status="Available and Idle";

description="Printer Cover";status="doorClosed";

in the following we can see the whole Information can be gathered about printers with npadmin:

Model and vendor Location and contact information, Network configuration, Memory and disk usage Max and min paper size Engine speed Duplexer installed Printer status Printer languages Marker technology Page count Minimum margins Size, capacity and level of paper trays Toner levels Alert conditions Resolution Display information Cover pages on/off.

Appendix D

Our codes for Context module (Presence Agent) for SER. Notice that the codes are not complete (for example there is no function for removing expired Subscribes, or Handle_Subscription function didn't define separately and I put it in the Module definition part just as an example).

```
/* Context Module */

#include "../../mem/shm_mem.h"
#include "../../mem/mem.h"
#include "../../fifo_server.h"
#include "../../usr_avp.h"
#include "../../parser/parse_uri.h"
#include "../../parser/parse_from.h"
#include "../../parser/parse_content.h"
#include "../../parser/parse_disposition.h"
#include "../../db/db.h"
#include "../../sr_module.h"
#include "../../str.h"
#include "../../msg_translator.h"
#include "../../data_lump_rpl.h"
#include "../../dprint.h"
#include "../../error.h"
#include "../../ut.h"
#include "../../globals.h"
#include "../../trim.h"
#include "../../parser/parse_event.h"
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>


#define Max_UDP_Buffer_size 4096
```

```
/* Module definition*/
MODULE_VERSION

static int Handle_Subscription(char *msg);
static int mod_init(void);
char* str_param;
int int_param;


static cmd_export_t cmds[]={
	{"context", Handle_Subscription, 1, 0, REQUEST_ROUTE},
	{0, 0, 0, 0, 0}
};

static param_export_t params[]={
	{"str_param", STR_PARAM, &str_param},
	{"int_param", INT_PARAM, &int_param},
	{0,0,0}
};

struct module_exports exports = {
	"context",
	cmds,		/* Exported functions */
	params,		/* Exported parameters */
	mod_init,	/* module initialization function */
	0,		/* response function*/
	0,		/* destroy function */
	0,		/* oncancel function */
	0		/* per-child init function */
};

static int mod_init(void)
{
	fprintf(stderr, "context - initializing\n");
	return 0;
}

static int print_f(struct sip_msg* msg, char* str, char* str2)
{
	/*we registered only 1 param, so we ignore str2*/
	printf("%s\n",str);
	return 1;
}
/* Print Module */
const char* pars1 = "SUBSCRIBE";
const char* pars2 = "SIP/2.0 200 OK";
```

```
const char* pars3 = "NOTIFY";
char mesg[Max_UDP_Buffer_size];
void ClearBuffer(char *msg, int size)
{ int j; /* index */
  for(j=0; j<size; j++)
 {
     msg[j]=0;
  }
}

/* This function reports the error and exits back to the shell */
static void bail (const char *on_what)
{
      fprintf(stderr, "%d: %s\n", errno, on_what);
      exit (1);
}


/* The function Creating a Random number/string with the length given */

char *CreateRandom(int Length)
{
      char *allowedChars,*randomstring;
      allowedChars = (char *)malloc(sizeof(char) * 80);
      randomstring = (char *)malloc(sizeof(char) * 50);
      int i,r;

      allowedChars=
"abcdefghi0123456789jkmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabcdefg
hi";

      srand( time(NULL) );
      for (i = 0; i < Length; i++)
            {
            r=rand();
            r= (r/100000000)*3;           /* Random number       */

            randomstring[i] = allowedChars[r];
            }
            return (randomstring);
}

/* The function return Subscribe state based on the expiration time
char *Substate()
{
      if ( Expired == 0 )               // user is not active more
            {
```

```
            Substate = "active";
            }
            else            // user is active
            {
            Substate = "pending";
            }
} */


/* Thread for getting messages from clients */
void *SendNotifyClientf(void *arg)
{
//    int j=0;
//    for(j;j<10;j++)
//    {
//    sleep(2);
      printf("Thread for sending Notify to Sun\n");
//    }
      return NULL;
}



/* Thread for getting messages from clients */
void *ReceiveFromClientf(void *arg)          //int argc, char **argv,
{
//   printf("Thread says hi!\n");

      int z, s, Expirei; /* s: Socket, z:bind the socket, u: used in snprintf to get the sender ip
address, Epire time in integer,Expire time dinamically callculated */
      socklen_t len_inet= Max_UDP_Buffer_size /* length */;
//    char *srvr_addr = NULL;
      char *Substate; //user current state based on expire time
      char Submsg[Max_UDP_Buffer_size]; /* Recv buffer, sender IP, Parsing sub msg */
      struct sockaddr_in adr_inet /* AF_INET */ , adr_clnt /* AF_INET */;
      char* result; /* this is the pointer to the xxx */


//    /* Use a address from the command line, otherwise, this program will default
//    to using the arbitrary address 130.237.15.238

//    if (argc >= 2)
//    {
//          /* Addr on cmdline:
//          srvr_addr = argv[1];
//    }
//    else
```

```
//    {
//          /* Use default address:
//          srvr_addr = "130.237.15.238";
//    }

      /* Create a UDP socket to use */
      s = socket (AF_INET, SOCK_DGRAM, 0);
      if (s == -1)
            bail ("socket()");

      /* Create a socket address, for use with bind(2) */
//    memset (&adr_inet, 0, sizeof adr_inet);
      bzero(&adr_inet,sizeof(adr_inet));
      adr_inet.sin_family = AF_INET;
      adr_inet.sin_port = htons (5060);
      adr_inet.sin_addr.s_addr = htonl(INADDR_ANY);
      if (adr_inet.sin_addr.s_addr == INADDR_NONE)
            bail ("bad address.");
      len_inet = sizeof adr_inet;

/* Bind a address to socket, so client programs can contact this server */
      z = bind (s, (struct sockaddr *) &adr_inet, len_inet);
      if (z == -1)
            bail ("bind()");

/* Now wait for requests */
      for (;;)
      {
            /* Block until the program receives a datagram at our address & port */
            ClearBuffer(mesg,Max_UDP_Buffer_size);                  //Clear the buffer
            len_inet = sizeof adr_clnt;
//          printf( "message before getting: : '%s' \n", mesg);
            z = recvfrom (s,mesg,   /* Receiving buffer */
Max_UDP_Buffer_size,MSG_WAITALL,
                        (struct sockaddr *) &adr_clnt,    /* Addr */
                         &len_inet);  /* Addr len, in & out */

            /* Find the message type, if it is subscribe or not */
            result = strstr( mesg, pars1 );
            if( result == NULL )     /* It is not a Subscribe message*/
            {
                  char* result2 = strstr( mesg, pars2 );
                  if( result2 == NULL ) /* it is not a Subscribe or Ok messages*/
                        {
                              char* result3 = strstr( mesg, pars3 );
                              if( result3 == NULL ) /*it is not Subscribe,Notify,Ok*/
```

```
                    {
                        ClearBuffer(mesg,Max_UDP_Buffer_size);
                        char * mesgback = "WhoAreYou";
                        sendto (s,mesgback, strlen(mesgback),0,(struct sockaddr *)
                        &adr_clnt,len_inet);
                    }
                    else  //it is a NOTIFY msg
                    {
                    printf( "NOTIFY message : '%s' \n", mesg );
                    ClearBuffer(mesg,Max_UDP_Buffer_size);
                    }
                }
            else /* It is a Ok messages */
                {
                printf( "Ok message send from Client: %s \n", mesg);
                ClearBuffer(mesg,Max_UDP_Buffer_size);
                }
        }

        else /*It is a Subscribe message*/
        {
        ClearBuffer(Submsg,Max_UDP_Buffer_size);
        strcpy(Submsg,mesg);
        ClearBuffer(mesg,Max_UDP_Buffer_size);
        printf("Subscribe message :%s\n", Submsg);    /* Display the received message */

            /* Parsing and extracting Subscribing message information */

            char Watcherstring[Max_UDP_Buffer_size];   //Extracting Watcherhost ,
            server IP address
            strcpy(Watcherstring, Submsg);
            char *Watcherhost=strtok(Watcherstring, " ");
            Watcherhost=strtok(NULL, " ");        //Watcherhost value
//          printf( "Watcherhost: '%s'\n", Watcherhost);

            char Branchstring[Max_UDP_Buffer_size];     //Extracting Branch
            strcpy(Branchstring, Submsg);
            char *branch=strtok(Branchstring, "=");
            branch=strtok(NULL, "\r");        //branch value
//          printf( "branch: '%s'\n", branch);

            char Tostring[Max_UDP_Buffer_size];     //Extracting To IP address, receiver
            strcpy(Tostring, Submsg);
            char * To=strtok(Tostring, "<");
            To=strtok(NULL, ">");       //To value
//          printf( "To: '%s'\n", To);
```

```
            char Fromstring[Max_UDP_Buffer_size]; //Extracting From IP address, Sender
            strcpy(Fromstring, Submsg);
            char* ParsFrom = "From";
            char* ResultFrom = strstr( Fromstring, ParsFrom );
            char  From2string[Max_UDP_Buffer_size];
            strcpy(From2string , ResultFrom);     //Header after From Word stored in
            ResultFrom
            char *From=strtok(From2string, "<");
            From=strtok(NULL, ">");          //From value
//          printf( "From: '%s'\n", From);

            char FromTagstring[Max_UDP_Buffer_size];  //Extracting From Tag
            strcpy(FromTagstring, ResultFrom);
            char* ParsFromTag = "tag";
            char* ResultFromTag = strstr( FromTagstring, ParsFromTag );
            char *FromTag=strtok(ResultFromTag, "=");
            FromTag=strtok(NULL, "\r");          //From Tag value
//          printf( "Tag (From): '%s'\n", FromTag);

            char Callidstring[Max_UDP_Buffer_size];       //Extracting Callid
            strcpy(Callidstring, ResultFrom);
            char* ParsCallid = "Call-ID";
            char* ResultCallid = strstr( Callidstring, ParsCallid );
            char *Callid=strtok(ResultCallid, " ");
            Callid=strtok(NULL, "\r");         //Call-ID value
//          printf( "Call-ID: '%s'\n", Callid);

            char CSeqstring[Max_UDP_Buffer_size]; //Extracting CSeq
            strcpy(CSeqstring, ResultFrom);
            char* ParsCSeq = "CSeq";
            char* ResultCSeq = strstr( CSeqstring, ParsCSeq );
            char *CSeq=strtok(ResultCSeq, " ");
            CSeq=strtok(NULL, "\r");          //CSeq value
//          printf( "CSeq: '%s'\n", CSeq);

            char Expirestring[Max_UDP_Buffer_size];      //Extracting Expires time
            strcpy(Expirestring, ResultFrom);
            char* ParsExpire = "Expires";
            char* ResultExpire = strstr( Expirestring, ParsExpire);
            char *Expire=strtok(ResultExpire, " ");
            Expire=strtok(NULL, "\r");        //Expires time value
            Expirei = atoi( Expire );
//          printf( "Expires: '%s'\n", Expire);

            char Contactstring[Max_UDP_Buffer_size];    //Extracting Contact
            strcpy(Contactstring, ResultFrom);
            char* ParsContact = "Contact";
```

```
                char* ResultContact = strstr( Contactstring, ParsContact);
                char *Contact=strtok(ResultContact, " ");
                Contact=strtok(NULL, "\r");             //Contact value
//              printf( "Contact: '%s'\n", Contact);

                char ContentLengthstring[Max_UDP_Buffer_size];//Extracting Content-Length
                strcpy(ContentLengthstring, ResultFrom);
                char* ParsContentLength = "Content-Length";
                char* ResultContentLength = strstr( ContentLengthstring, ParsContentLength);
                char *ContentLength=strtok(ResultContentLength, " ");
                ContentLength=strtok(NULL, "\r");          //Content-Length value
//              printf( "Content-Length: '%s'\n", ContentLength);

                char Eventstring[Max_UDP_Buffer_size];//Extracting Event
                strcpy(Eventstring, ResultFrom);
                char* ParsEvent = "Event";
                char* ResultEvent = strstr( Eventstring, ParsEvent);
                char *Event=strtok(ResultEvent, " ");
                Event=strtok(NULL, "\r");          //Event value
//              printf( "Event: '%s'\n", Event);


                /* End of Parsing Subscribe messages  */

                /* Creating a random Tag (To) */
                char  *TagTo;
                TagTo=CreateRandom(4);
//              printf( "Tag (To): %s\n", TagTo);

                /* creating Ok Header - values has gotten from Subscribe message  */
                char OkHeader[Max_UDP_Buffer_size];
                sprintf(OkHeader, "SIP/2.0 200 OK\r\nVia: SIP/2.0/UDP
%s;branch=%s;received=%s\r\nTo: <%s>;tag=%s\r\nFrom: <%s>;tag= %s\r\nCall-ID:
%s\r\nCSeq: %s\r\nExpires: %s\r\nContact: %s\r\nContent-Length: %s\r\n\r\n", Watcherhost ,
branch , Watcherhost , To , TagTo, From , FromTag , Callid , CSeq , Expire , Contact ,
ContentLength);

                printf( "OkHeader: '%s'\n", OkHeader );


                /*Send back a Ok message to the client program, for Subscribe meesages*/
                adr_clnt.sin_port = htons (5060);/* assign the port number for sending
messages */
                sendto (s,OkHeader, strlen(OkHeader),0,(struct sockaddr *)
&adr_clnt,len_inet);
                OkHeader[0]='\0';
```

```
                /*Creating Notify messages after Ok msg to client to inform current location */

                /* Creating a random branch (Notify_Ok) */
                char *branchNotify;
                branchNotify=CreateRandom(14);
//              printf( "branchNotify: %s\n", branchNotify);

                /* Creating a random CSeqNotify */
                int  CSeqNotify;
                srand( time(NULL) );
                CSeqNotify=rand();
                CSeqNotify= (CSeqNotify/1000000);       /* CSeqNotify       */
//              printf( "CSeqNotify: %d\n", CSeqNotify);

                /* Callculating Expires time */
//              Substated();          //achieving user state based on Expire time
                if ( Expirei == 0 ) /* user is not active more*/
                      {
                      Substate = "pending";

                      }
                else /* user is active */
                      {
                      Substate = "active";
                      }


                /* Extracting user Status: Active/disabled based on Expire time */
//              Substate();           //achieving Expire time from Expire time of Subscribtion
                Expirei=Expirei-1;
//              printf( "Expirei: %d\n", Expirei);

                char NotifyOkHeader[Max_UDP_Buffer_size];

                sprintf(NotifyOkHeader, "NOTIFY %s SIP/2.0\nVia: SIP/2.0/UDP
%s;branch=%s\nFrom: <%s>;tag=%s\nTo: <%s>;tag= %s\nCall-ID: %s\nEvent:
%s\nSubscription-State: %s;expires=%d\nMax-Forwards: 70\nCSeq: %d NOTIFY\nContact:
%s\nContent-Type: application/pidf+xml\nContent-Length: ...\n", From , To, branchNotify ,
To , TagTo, From , FromTag , Callid , Event , Substate , Expirei, CSeqNotify , Contact);

                printf( "NotifyOkHeader: '%s'\n", NotifyOkHeader);

                /*Send back a Notify message to the client program, after Ok for his Subs
msg*/
                usleep(100000);
```

```
                adr_clnt.sin_port = htons (5060);/* assign the port number for sending
messages */
                sendto (s,NotifyOkHeader, strlen(NotifyOkHeader),0,(struct sockaddr *)
&adr_clnt,len_inet);
                NotifyOkHeader[0]='\0';
            }

//          /* Process the request */
//          mesg[z] = 0;  /* null terminate */
//          if (!strcasecmp (mesg, "QUIT"))
//              break;     /* Quit server */

      }                 /*it is a for bracket of receiving message*/

/* Close the socket and exit */
      close (s);
//    return 0;

  return NULL;
}

/* Main function */
int main (void)
{
pthread_t ReceiveFromClient,SendNotifyClient;

      if ( pthread_create( &ReceiveFromClient, NULL, ReceiveFromClientf, NULL) )
      {
      printf("error creating thread.");
      abort();
      }

      if ( pthread_create( &SendNotifyClient, NULL, SendNotifyClientf, NULL) )
      {
      printf("error creating thread.");
      abort();
      }
      if ( pthread_join ( ReceiveFromClient, NULL ) )
      {
      printf("error joining thread.");
      abort();
      }
```

Appendix E

```
# Ser.cfg file
# configured as a Presence server by loading PA module
# @ Mohammad Zarifi
debug=9                 # debug level (cmd line: -ddddddddd)

check_via=no            # (cmd. line: -v)
dns=no                  # (cmd. line: -r)
rev_dns=no              # (cmd. line: -R)
port=5060
listen=130.237.15.238   # the server address (it will listen only on this interface)
children=2
#alias="Wireless-kth.com"

# ------------------ module loading ----------------------------------

# Uncomment this if you want to use SQL database
loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/avp.so"
loadmodule "/usr/local/lib/ser/modules/avpops.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/textops.so"
loadmodule "/usr/local/lib/ser/modules/mysql.so"

loadmodule "/usr/local/lib/ser/modules/dialog.so"
loadmodule "/usr/local/lib/ser/modules/rls.so"
loadmodule "/usr/local/lib/ser/modules/pa.so"
loadmodule "/usr/local/lib/ser/modules/presence_b2b.so"
loadmodule "/usr/local/lib/ser/modules/uri.so"
loadmodule "/usr/local/lib/ser/modules/uri_db.so"
loadmodule "/usr/local/lib/ser/modules/domain.so"
loadmodule "/usr/local/lib/ser/modules/fifo.so"
loadmodule "/usr/local/lib/ser/modules/xmlrpc.so"
loadmodule "/usr/local/lib/ser/modules/xlog.so"

# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"
loadmodule "/usr/local/lib/ser/modules/msilo.so"
```

```
# ----------------- setting module-specific parameters ---------------

modparam("msilo","use_contact",0)
modparam("msilo","expire_time",7200)

# -- auth params --
# Uncomment if you are using auth module
modparam("auth_db", "calculate_ha1", yes)
# If you set "calculate_ha1" parameter to yes (which true in this config),
# uncomment also the following parameter)
modparam("auth_db", "password_column", "password")

# -- rr params --
# add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)

modparam("rls", "min_expiration", 200)
modparam("rls", "max_expiration", 300)
modparam("rls", "default_expiration", 300)
modparam("rls", "auth", "none")
modparam("rls", "xcap_root", "http://localhost/xcap")
modparam("rls", "reduce_xcap_needs", 1)
modparam("rls", "db_mode", 1)
modparam("rls", "db_url", "mysql://ser:heslo@localhost:3306/ser")

modparam("pa", "use_db", 1)
# allow storing authorization requests for offline users into database
modparam("pa", "use_offline_winfo", 1)
# how often try to remove old stored authorization requests
modparam("pa", "offline_winfo_timer", 600)
# how long stored authorization requests live
modparam("pa", "offline_winfo_expiration", 600)
modparam("pa", "db_url", "mysql://ser:heslo@localhost:3306/ser")
# mode of PA authorization: none, implicit or xcap
modparam("pa", "auth", "none")
modparam("pa", "auth_xcap_root", "http://localhost/xcap")
# do not authorize watcherinfo subscriptions
modparam("pa", "winfo_auth", "none")
# use only published information if set to 0
modparam("pa", "use_callbacks", 1)
# dont accept internal subscriptions from RLS, ...
modparam("pa", "accept_internal_subscriptions", 0)
# maximum value of Expires for subscriptions
modparam("pa", "max_subscription_expiration", 600)
# maximum value of Expires for publications
modparam("pa", "max_publish_expiration", 120)
```

```
# how often test if something changes and send NOTIFY
modparam("pa", "timer_interval", 2)

# route for generated SUBSCRIBE requests for presence
modparam("presence_b2b", "presence_route", "<sip:127.0.0.1;transport=tcp;lr>")
# waiting time from error to new attepmt about SUBSCRIBE
modparam("presence_b2b", "on_error_retry_time", 60)
# how long wait for NOTIFY with Subscription-Status=terminated after unsubscribe
modparam("presence_b2b", "wait_for_term_notify", 33)
# how long before expiration send renewal SUBSCRIBE request
modparam("presence_b2b", "resubscribe_delta", 30)
# minimal time to send renewal SUBSCRIBE request from receiving previous response
modparam("presence_b2b", "min_resubscribe_time", 60)
# default expiration timeout
modparam("presence_b2b", "default_expiration", 3600)
# process internal subscriptions to presence events
modparam("presence_b2b", "handle_presence_subscriptions", 1)

modparam("usrloc", "db_mode", 1)
modparam("domain", "db_mode", 1)
modparam("domain|uri_db|acc|auth_db|usrloc|msilo", "db_url", "mysql://ser:heslo@localhost
:3306/ser")

modparam("fifo", "fifo_file", "/tmp/ser_fifo")

# ------------------------  request routing logic -------------------

# main routing logic

route{

   # XML RPC
   if (method == "POST" ||  method == "GET") {
      create_via();
      dispatch_rpc();
      break;
   }

   # initial sanity checks -- messages with
   # max_forwards==0, or excessively long requests
   if (!mf_process_maxfwd_header("10")) {
      sl_send_reply("483","Too Many Hops");
      break;
   };
   if (msg:len >=  max_len ) {
      sl_send_reply("513", "Message too big");
      break;
```

```
};

# we record-route all messages -- to make sure that
# subsequent messages will go through our proxy; that's
# particularly good if upstream and downstream entities
# use different transport protocol
if (!method=="REGISTER") record_route();

# subsequent messages withing a dialog should take the
# path determined by record-routing
if (loose_route()) {
    # mark routing logic in request
    append_hf("P-hint: rr-enforced\r\n");
    route(1);
    break;
};

# if the request is for other domain use UsrLoc
# (in case, it does not work, use the following command
# with proper names and addresses in it)
if (uri=~" 130.237.15.238") {

    if (!lookup_domain("To")) {
        xlog("L_ERR", "Unknown domain to: %tu from: %fu\n");
        route(1);
        break;
    }

        if (method=="SUBSCRIBE") {
            log(1,"Subscribe\n");
            if (t_newtran()) {
                    log(1,"Register\n");
                    handle_subscription("registrar");
                log(1,"Done\n");
                };
            break;
           };
        if (method=="PUBLISH") {
            log(1,"Publish\n");
             if (!t_newtran()) {
                 log(1,"newtran error\n");
                sl_reply_error();
                };
            handle_publish("registrar");
                log(1,"publish handled\n");
            break;
           };
```

```
# get user (common for all other messages than SUBSCRIBE)
if (!lookup_user("To")) {
    # log(1, "Unknown user - message should be forwarded?");
    # break;
    append_hf("P-hint: unknown user\r\n");
    route(1);
    break;
}
if (method=="NOTIFY") {
     if (!t_newtran()) {
        log(1, "newtran error\n");
        sl_reply_error();
       break;
     };
    # handle notification sent in internal subscriptions (presence_b2b)
     if (!handle_notify()) {
         t_reply("481", "Unable to handle notification");
     }
     break;
 };
if (method=="MESSAGE") {
       if (authorize_message("http://localhost/xcap")) {
       # use usrloc for delivery
        if (lookup("location")) {
            log(1, "Delivering MESSAGE using usrloc\n");
            t_on_failure("1");
            if (!t_relay()) {
                sl_reply_error();
            }
            break;
        }
        else {
            # store messages for offline user
            xlog("L_ERR", "MSILO: storing MESSAGE for %tu\n");

            if (!t_newtran()) {
               log(1, "newtran error\n");
               sl_reply_error();
               break;
            };

            # store only text messages NOT isComposing... !
            if (search("^(Content-Type|c):.*application/im-iscomposing\+xml.*")) {
                log(1, "it is only isComposing message - ignored\n");
                t_reply("202", "Ignored");
                break;
            }
```

```
                if (m_store("0", "sip:127.0.0.1")) {
#                 log(1, "MSILO: offline message stored\n");
                if (!t_reply("202", "Accepted")) {
                    sl_reply_error();
                };
            } else {
                log(1, "MSILO: error storing offline message\n");
                if (!t_reply("503", "Service Unavailable")) {
                    sl_reply_error();
                };
            };
            break;
        }
        break;
    }
    else {
        # log(1, "unauthorized message\n");
        sl_reply("403", "Forbidden");
    }
    break;
  }
  if (method=="REGISTER") {
    # uncomment this if you want to authenticate REGISTER request
    if (!www_authenticate(" 130.237.15.238", "credentials")) {
        www_challenge( "130.237.15.238", "0");
        break;
    };
    save("location");
    # dump stored messages - route it through myself (otherwise routed via DNS!)
    if (m_dump("sip: 127.0.0.1")) {
        xlog("L_ERR", "MSILO: offline messages for %fu dumped\n");
    }
    break;
  };
  # native SIP destinations are handled using our USRLOC DB
  if (!lookup("location")) {
    sl_send_reply("404", "Not Found");
    break;
  };
};
#   append_hf("P-hint: usrloc applied\r\n");
  route(1);
}
route[1]
{
  # send it out now; use stateful forwarding as it works reliably
  # even for UDP2TCP
```

```
    if (!t_relay()) {
        sl_reply_error();
    };
}
failure_route[1] {
    # forwarding failed -- check if the request was a MESSAGE
    if (!method=="MESSAGE") { break; };
    log(1, "MSILO: MESSAGE forward failed - storing it\n");    # we have changed the R-URI
with the contact address, ignore it now
    if (m_store("0", "")) {
        t_reply("202", "Accepted");
    } else {
        log(1, "MSILO: offline message NOT stored\n");
        t_reply("503", "Service Unavailable");
    };
}
        if ( pthread_join ( SendNotifyClient, NULL ) )
        {
        printf("error joining thread.");
        abort();
        }
exit(0);
}
```

Appendix F

In this appendix you can see the edited source codes of SER, in order to make its performance appropriate as our server. Notice that in every file I just put the part of the source code, which is edited by me or related to our application. Therefore if you want to have the proper functionality of codes you should copy the rest of codes from original files. You can find the complete and original source code of each file by downloading the SER from iptel.org website.

F.1. Parse_event.h

```
#ifndef PARSE_EVENT_H
#define PARSE_EVENT_H

#include "../str.h"
#include "hf.h"

#define EVENT_OTHER         0
#define EVENT_PRESENCE      1
#define EVENT_PRESENCE_WINFO 2
#define EVENT_SIP_PROFILE   3
#define EVENT_XCAP_CHANGE   4

#define EVENT_LOCATION      5

#define EVENT_ROOMA         6
#define EVENT_ROOMB         7
#define EVENT_ROOMC         8

typedef struct event {
   str text;       /* Original string representation */
   int parsed;     /* Parsed variant */
} event_t;
```

F.2. Parse_event.c

```
#include "parse_event.h"
#include "../mem/mem.h"    /* pkg_malloc, pkg_free */
#include "../dprint.h"
#include <string.h>        /* memset */
#include "../trim.h"       /* trim_leading */
#include <stdio.h>         /* printf */
#include "../ut.h"


#define PRES_STR "presence"
#define PRES_STR_LEN 8

/* Mohammad's code */
```

```
#define LOC_STR "location"
#define LOC_STR_LEN 8

#define ROMA_STR "roomA"
#define ROMA_STR_LEN 5

#define ROMB_STR "roomB"
#define ROMB_STR_LEN 5

#define ROMC_STR "roomC"
#define ROMC_STR_LEN 5

/* Mohammad's code */
#define PRES_WINFO_STR "presence.winfo"
#define PRES_WINFO_STR_LEN 14

#define PRES_XCAP_CHANGE_STR "xcap-change"
#define PRES_XCAP_CHANGE_STR_LEN 11

#define PRES_SIP_PROFILE_STR "sip-profile"
#define PRES_SIP_PROFILE_STR_LEN 11

static inline char* skip_token(char* _b, int _l)
{
    int i = 0;

    for(i = 0; i < _l; i++) {
        switch(_b[i]) {
        case ' ':
        case '\r':
        case '\n':
        case '\t':
        case ';':
            return _b + i;
        }
    }

    return _b + _l;
}

static inline int event_parser(char* _s, int _l, event_t* _e)
{
    str tmp;
    char* end;

    tmp.s = _s;
    tmp.len = _l;

    trim_leading(&tmp);

    if (tmp.len == 0) {
```

```
    LOG(L_ERR, "event_parser(): Empty body\n");
    return -1;
}

_e->text.s = tmp.s;

end = skip_token(tmp.s, tmp.len);

_e-> text.len = end - tmp.s;

if ((_e->text.len == PRES_STR_LEN) &&
    !strncasecmp(PRES_STR, tmp.s, _e->text.len)) {
    _e->parsed = EVENT_PRESENCE;
}

/* Mohammad's code */
else if ((_e->text.len == LOC_STR_LEN) &&
    !strncasecmp(LOC_STR, tmp.s, _e->text.len)) {
    _e->parsed = EVENT_LOCATION;
}

else if ((_e->text.len == ROMA_STR_LEN) &&
    !strncasecmp(ROMA_STR, tmp.s, _e->text.len)) {
    _e->parsed = EVENT_ROOMA;
}

else if ((_e->text.len == ROMB_STR_LEN) &&
    !strncasecmp(ROMB_STR, tmp.s, _e-> text.len)) {
    _e->parsed = EVENT_ROOMB;
}

else if ((_e->text.len == ROMC_STR_LEN) &&
    !strncasecmp(ROMC_STR, tmp.s, _e->text.len)) {
    _e->parsed = EVENT_ROOMC;
}
/* Mohammad's code */

else if ((_e->text.len == PRES_XCAP_CHANGE_STR_LEN) &&
      !strncasecmp(PRES_XCAP_CHANGE_STR, tmp.s, _e->text.len)) {
    _e->parsed = EVENT_XCAP_CHANGE;
} else if ((_e->text.len == PRES_WINFO_STR_LEN) &&
      !strncasecmp(PRES_WINFO_STR, tmp.s, _e->text.len)) {
    _e->parsed = EVENT_PRESENCE_WINFO;
} else if ((_e->text.len == PRES_SIP_PROFILE_STR_LEN) &&
      !strncasecmp(PRES_SIP_PROFILE_STR, tmp.s, _e->text.len)) {
    _e->parsed = EVENT_SIP_PROFILE;
} else {
    _e->parsed = EVENT_OTHER;
}
```

```
    return 0;
}
```

F.3. Subscribe.c

```
#include <string.h>
#include <limits.h>
#include "../../str.h"
#include "../../dprint.h"
#include "../../mem/mem.h"
#include "../../parser/parse_uri.h"
#include "../../parser/parse_from.h"
#include "../../parser/parse_expires.h"
#include "../../parser/parse_event.h"
#include "../../parser/parse_content.h"
#include "../../data_lump_rpl.h"
#include "presentity.h"
#include "watcher.h"
#include "pstate.h"
#include "notify.h"
#include "paerrno.h"
#include "pdomain.h"
#include "pa_mod.h"
#include "ptime.h"
#include "reply.h"
#include "subscribe.h"
#include "auth.h"
#include <cds/sstr.h>
#include <cds/msg_queue.h>
#include <cds/logger.h>
#include <presence/utils.h>

#define DOCUMENT_TYPE "application/cpim-pidf+xml"
#define DOCUMENT_TYPE_L (sizeof(DOCUMENT_TYPE) - 1)

typedef struct {
    int event_type;
    int mimes[MAX_MIMES_NR];
} event_mimetypes_t;

static event_mimetypes_t event_package_mimetypes[] = {
    { EVENT_PRESENCE, {
            MIMETYPE(APPLICATION,PIDFXML),
/*          MIMETYPE(APPLICATION,XML_MSRTC_PIDF), */
/*          MIMETYPE(TEXT,XML_MSRTC_PIDF), */
            MIMETYPE(APPLICATION,CPIM_PIDFXML),
            MIMETYPE(APPLICATION,XPIDFXML),
            MIMETYPE(APPLICATION,LPIDFXML),
            0 } },

    /* Mohammad's code */
```

```
    { EVENT_LOCATION, {
          MIMETYPE(APPLICATION,PIDFXML),
          0 } },
    { EVENT_ROOMA, {
          MIMETYPE(APPLICATION,PIDFXML),
          0 } },
    { EVENT_ROOMB, {
          MIMETYPE(APPLICATION,PIDFXML),
          0 } },
    { EVENT_ROOMC, {
          MIMETYPE(APPLICATION,PIDFXML),
          0 } },


    { EVENT_PRESENCE_WINFO, {
          MIMETYPE(APPLICATION,WATCHERINFOXML),
          0 } },
/*    { EVENT_SIP_PROFILE, {
          MIMETYPE(MESSAGE,EXTERNAL_BODY),
          0 } }, */
/*    { EVENT_XCAP_CHANGE, { MIMETYPE(APPLICATION,WINFO+XML), 0 } }, */
    { -1, { 0 }},
};

...
```

F.4. Watcher.c

```
#include "paerrno.h"
#include "../../db/db.h"
#include "../../dprint.h"
#include "../../parser/parse_event.h"
#include "../../mem/shm_mem.h"
#include "../../trim.h"
#include "../../ut.h"
#include "pa_mod.h"
#include "common.h"
#include "watcher.h"
#include "presentity.h"
#include "auth.h"
#include " ptime.h"

str watcher_status_names[] = {
    [WS_PENDING] = STR_STATIC_INIT("pending"),
    [WS_ACTIVE] = STR_STATIC_INIT("active"),
    [WS_REJECTED] = STR_STATIC_INIT("rejected"),
    [WS_TERMINATED] = STR_STATIC_INIT("terminated"),
    [WS_PENDING_TERMINATED] = STR_STATIC_INIT("terminated"),
    STR_NULL
};
```

```
str watcher_event_names[] = {
    [WE_SUBSCRIBE]   = STR_STATIC_INIT("subscribe"),
    [WE_APPROVED]    = STR_STATIC_INIT("approved"),
    [WE_DEACTIVATED] = STR_STATIC_INIT("deactivated"),
    [WE_PROBATION]   = STR_STATIC_INIT("probation"),
    [WE_REJECTED]    = STR_STATIC_INIT("rejected"),
    [WE_TIMEOUT]     = STR_STATIC_INIT("timeout"),
    [WE_GIVEUP]      = STR_STATIC_INIT("giveup"),
    [WE_NORESOURCE]  = STR_STATIC_INIT("noresource"),
    STR_NULL
};

const char *event_package2str(int et) /* FIXME: change working with this to enum ?*/
{
   /* added due to incorrect package handling */
   switch (et) {
      case EVENT_PRESENCE: return "presence";
      case EVENT_PRESENCE_WINFO: return "presence.winfo";

      /*        Mohammad's code        */
      case EVENT_LOCATION: return "location";
      case EVENT_ROOMA: return "roomA";
      case EVENT_ROOMB: return "roomB";
      case EVENT_ROOMC: return "roomC";

      /*case EVENT_XCAP_CHANGE: return ...; */
      default: return "unknown";
   }
}

int str2event_package(const char *epname)
{
   /* work only with packages supported by PA! */
   if (strcmp(epname, "presence") == 0) return EVENT_PRESENCE;
   if (strcmp(epname, " presence.winfo") == 0) return EVENT_PRESENCE_WINFO;
   return -1; /* unsupported */
}

/* returns 0 if package supported by PA */
int verify_event_package(int et)
{
   switch (et) {
      case EVENT_PRESENCE: return 0;

      /* Code for location event from Mohammad */

      case EVENT_LOCATION: return 0;
      case EVENT_ROOMA: return 0;
      case EVENT_ROOMB: return 0;
      case EVENT_ROOMC: return 0;
```

```
        case EVENT_PRESENCE_WINFO:
            if (watcherinfo_notify) return 0;
            else return -1;
        default: return -1;
    }
}

watcher_status_t watcher_status_from_string(str *wsname)
{
    int i;
    for (i = 0; watcher_status_names[i].len; i++) {
     if (str_strcasecmp(wsname, &watcher_status_names[i]) == 0) {
         return i;
     }
    }
    return 0;
}

watcher_event_t watcher_event_from_string(str *wename)
{
    int i;
    for (i = 0; watcher_event_names[i].len; i++) {
     if (str_strcasecmp(wename, &watcher_event_names[i]) == 0) {
         return i;
     }
    }
    return 0;
}
```

F.5. Publish.c

```
#include<stdio.h>
#include <string.h>
#include <stdlib.h>
#include "../../str.h"
#include "../../dprint.h"
#include "../../mem/mem.h"
#include "../../parser/parse_uri.h"
#include "../../parser/parse_from.h"
#include "../../parser/contact/parse_contact.h"
#include "../../parser/parse_expires.h"
#include "../../parser/parse_event.h"
#include " dlist.h"
#include "presentity.h"
#include "watcher.h"
#include "pstate.h"
#include "notify.h"
#include "paerrno.h"
#include "pdomain.h"
#include "pa_mod.h"
```

```
#include "ptime.h"
#include "reply.h"
#include "subscribe.h"
#include "publish.h"
#include "common.h"
#include "../../data_lump_rpl.h"
#include "../../parser/parse_sipifmatch.h"

#include <libxml/parser.h>
#include <libxml/xpath.h>

#include <presence/pidf.h>
#include <cds/logger.h>

// added by Mohammad ///
#include "offline_winfo.h"
#include "winfo_doc.h"
#include "message.h"
#include <cds/sstr.h>
// added by Mohammad ///

.
.
.

presence_tuple_t *presence_tuple_info2pa(presence_tuple_info_t *i, str *etag, time_t expires)
{
   presence_tuple_t *t = NULL;
   presence_note_t *n, *nn;

   int res;

   res = new_presence_tuple(&i->contact, expires, &t, 1);
   if (res != 0) return NULL;
   /* ID for the tuple is newly generated ! */
   t->priority = i->priority;
   switch (i->status) {
      case presence_tuple_open: t->state = PS_ONLINE; break;
      case presence_tuple_closed: t->state = PS_OFFLINE; break;
   }
   str_dup(&t->etag, etag);
   str_dup(&t->published_id, &i->id); /* store published tuple ID - used on update */

   //added by Mohammad for location tag, here added from tuple_info to tuple structure (-;
   str_clear(&t->description);
   t->description.len = i->description.len;
   LOG(L_DBG, "\n description in Publish file: [ %s ]\n", i->description.s);
   str_dup(&t->description, &i->description);

   str_clear(&t->floor);
   t->floor.len = i->floor.len;
```

```
    str_dup(&t->floor, &i->floor);

    str_clear(&t->room);
    t->room.len = i->room.len;
    str_dup(&t->room, &i->room);

    str_clear(&t->longtitude);
    t->longtitude.len = i->longtitude.len;
    str_dup(&t->longtitude, &i->longtitude);

    str_clear(&t->latitude);
    t->latitude.len = i->latitude.len;
    str_dup(&t->latitude, &i->latitude);

    str_clear(&t->height);
    t->height.len = i->height.len;
    str_dup(&t->height, &i->height); /* store published tuple ID - used on update */
    //added by Mohammad for location tag


    /* add notes for tuple */
    n = i->first_note;
    while (n) {
        nn = create_presence_note(&n->value, &n->lang);
        if (nn) add_tuple_note_no_wb(t, nn);
        n = n->next;
    }

    return t;
}
.
.
.
static int publish_presentity(struct sip_msg* _m, struct pdomain* _d, struct presentity* presentity)
{
    event_t *parsed_event = NULL;
    int event_package = EVENT_OTHER;
    str callid = STR_STATIC_INIT("???");
    int changed = 0; /* temporarily */
    int res;
    char* resultevent;
    presence_tuple_t* tupleresult;


    if (_m->event)
        parsed_event = (event_t *)_m->event->parsed;
    if (parsed_event)
        event_package = parsed_event->parsed;


    switch (event_package) {
```

```
case EVENT_PRESENCE:

LOG(L_DBG, "\n\n\n Tuple id =%d -1-\n\n\n", event_package);

   resultevent = (char *)malloc(sizeof(char) * 50);
   resultevent= strtok(_m->event-> name.s, " ");
   resultevent = strtok(NULL, "\r\n");

   //Mohammad Code//
   tupleresult = (presence_tuple_t*)mem_alloc(100);

   tupleresult-> event.s = resultevent;

   //LOG(L_DBG, "publish_presentity: event_package= [ %s ] -1-\n", t->published_id.s);


   if (!use_db) return 0;
   db_key_t cols[1];
   db_val_t vals[1];
   int n_updates = 0;

   cols[n_updates] = "presid";
   vals[n_updates].type = DB_INT;
   vals[n_updates].nul = 0;
   vals[n_updates].val.int_val = presentity->presid;
   n_updates++;

   cols[n_updates] = "event";
   vals[n_updates].type = DB_STRING;
   vals[n_updates].nul = 0;
   vals[n_updates].val.string_val = tupleresult-> event.s;

LOG(L_DBG, "\n\n\nbefore inserting to event tale: event_package= [ %s ] \n\n\n", tupleresult-
>event.s);

   if (pa_dbf.use_table(pa_db, presentity_event_table) < 0) {
      LOG(L_ERR, "db_add_event_element: Error in use_table\n");
      return -1;
   }

   if (pa_dbf.insert(pa_db, cols, vals, 2) < 0) {
      LOG(L_ERR, "db_add_event_element: Can't insert record\n");
      return -1;
   }


res = publish_presence(_m, presentity);
//presentity->presid++;       //Mohammad  (-;
   break;

/*Mohammad's code !*/
```

```
case EVENT_LOCATION:

   resultevent = (char *)malloc(sizeof(char) * 50);
   resultevent= strtok(_m->event->name.s, " ");
   resultevent = strtok(NULL, "\r\n");

   tupleresult = (presence_tuple_t*)mem_alloc(100);

   tupleresult->event.s = resultevent;


   if (!use_db) return 0;
   db_key_t cols2[1];
   db_val_t vals2[1];
   int n_updates2 = 0;

   cols2[n_updates2] = "presid";
   vals2[n_updates2].type = DB_INT;
   vals2[n_updates2].nul = 0;
   vals2[n_updates2].val.int_val = presentity->presid;
   n_updates2++;

   cols2[n_updates2] = "event";
   vals2[n_updates2].type = DB_STRING;
   vals2[n_updates2].nul = 0;
   vals2[n_updates2].val.string_val = tupleresult->event.s;

   if (pa_dbf.use_table(pa_db, presentity_event_table) < 0) {
      LOG(L_ERR, "db_add_event_element: Error in use_table\n");
      return -1;
   }

   if (pa_dbf.insert(pa_db, cols2, vals2, 2) < 0) {
      LOG(L_ERR, "db_add_event_element: Can't insert record\n");
      return -1;
   }



   res = publish_presence(_m, presentity);
   //presentity->presid++;       //Mohammad  (-;
   break;

   case EVENT_ROOMA:

   resultevent = (char *)malloc(sizeof(char) * 50);
   resultevent= strtok(_m->event->name.s, " ");
   resultevent = strtok(NULL, "\r\n");

   tupleresult = (presence_tuple_t*)mem_alloc(100);
```

```
tupleresult->event.s = resultevent;


if (!use_db) return 0;
db_key_t cols3[1];
db_val_t vals3[1];
int n_updates3 = 0;

cols3[n_updates3] = "presid";
vals3[n_updates3].type = DB_INT;
vals3[n_updates3].nul = 0;
vals3[n_updates3].val.int_val = presentity->presid;
n_updates3++;

cols3[n_updates3] = "event";
vals3[n_updates3].type = DB_STRING;
vals3[n_updates3].nul = 0;
vals3[n_updates3].val.string_val = tupleresult->event.s;

LOG(L_DBG, "\n\n\nbefore inserting to event tale: event_package= [ %s ] \n\n\n", tupleresult->event.s);

if (pa_dbf.use_table(pa_db, presentity_event_table) < 0) {
   LOG(L_ERR, "db_add_event_element: Error in use_table\n");
   return -1;
}

if (pa_dbf.insert(pa_db, cols3, vals3, 2) < 0) {
   LOG(L_ERR, "db_add_event_element: Can't insert record\n");
   return -1;
}

res = publish_presence(_m, presentity);
//presentity->presid++;       //Mohammad  (-;
break;


case EVENT_ROOMB:

resultevent = (char *)malloc(sizeof(char) * 50);
resultevent= strtok(_m->event->name.s, " ");
resultevent = strtok(NULL, "\r\n");

tupleresult = (presence_tuple_t*)mem_alloc(100);

tupleresult->event.s = resultevent;


if (!use_db) return 0;
db_key_t cols4[1];
```

```
db_val_t vals4[1];
int n_updates4 = 0;

cols4[n_updates4] = "presid";
vals4[n_updates4].type = DB_INT;
vals4[n_updates4].nul = 0;
vals4[n_updates4].val.int_val = presentity->presid;
n_updates4++;

cols4[n_updates4] = "event";
vals4[n_updates4].type = DB_STRING;
vals4[n_updates4].nul = 0;
vals4[n_updates4].val.string_val = tupleresult->event.s;

if (pa_dbf.use_table(pa_db, presentity_event_table) < 0) {
   LOG(L_ERR, "db_add_event_element: Error in use_table\n");
   return -1;
}

if (pa_dbf.insert(pa_db, cols4, vals4, 2) < 0) {
   LOG(L_ERR, "db_add_event_element: Can't insert record\n");
   return -1;
}

res = publish_presence(_m, presentity);
//presentity->presid++;       //Mohammad  (-;
break;

case EVENT_ROOMC:

resultevent = (char *)malloc(sizeof(char) * 50);
resultevent= strtok(_m->event->name.s, " ");
resultevent = strtok(NULL, "\r\n");

tupleresult = (presence_tuple_t*)mem_alloc(100);

tupleresult->event.s = resultevent;


if (!use_db) return 0;
db_key_t cols5[1];
db_val_t vals5[1];
int n_updates5 = 0;

cols5[n_updates5] = "presid";
vals5[n_updates5].type = DB_INT;
vals5[n_updates5].nul = 0;
vals5[n_updates5].val.int_val = presentity->presid;
n_updates5++;

cols5[n_updates5] = "event";
```

```
        vals5[n_updates5].type = DB_STRING;
        vals5[n_updates5].nul = 0;
        vals5[n_updates5].val.string_val = tupleresult->event.s;

        if (pa_dbf.use_table(pa_db, presentity_event_table) < 0) {
           LOG(L_ERR, "db_add_event_element: Error in use_table\n");
           return -1;
        }

        if (pa_dbf.insert(pa_db, cols5, vals5, 2) < 0) {
           LOG(L_ERR, "db_add_event_element: Can't insert record\n");
           return -1;
        }

        res = publish_presence(_m, presentity);
        //presentity->presid++;       //Mohammad  (-;
        break;


        //Mohammad Code//


      /*********************/

      case EVENT_XCAP_CHANGE:
        /* FIXME: throw it out - it is not presence related, it is XCAP */
        /* FIXME: add headers Expires and SIP-ETag */
        res = publish_presentity_xcap_change(_m, _d, presentity, &changed);
        break;
      default:
        if (_m->callid)    callid = _m->callid->body;
        LOG(L_WARN, "publish_presentity: no handler for event_package=%d"
              " callid=%.*s\n", event_package, callid.len, ZSW(callid.s));
        paerrno = PA_EVENT_UNSUPP;
        res = -1;
   }

   return res;
}

/*
 * Handle a publish Request
 */

int handle_publish(struct sip_msg* _m, char* _domain, char* _s2)
{
   struct pdomain* d;
   struct presentity *p;
   str p_uri = STR_NULL;
   str uid = STR_NULL;
```

```
    get_act_time();
    paerrno = PA_OK;

    DBG("handle_publish(): init!");

    if (parse_publish_hfs(_m) < 0) {
        LOG(L_ERR, "handle_publish(): Error while parsing message header\n");
        goto error;
    }

#if 0
    if (check_message(_m) < 0) {
        LOG(L_ERR, "handle_publish(): Error while checking message\n");
        goto error;
    }
    LOG(L_ERR, "handle_publish -1c-\n");
#endif
    LOG(L_DBG, "handle_publish entered\n");

    d = (struct pdomain*)_domain;

    if (get_pres_uri(_m, &p_uri) < 0 || p_uri.s == NULL || p_uri.len == 0) {
        LOG(L_ERR, "handle_publish(): Error while extracting presentity URI\n");
        goto error;
    }

    if (get_presentity_uid(&uid, _m) != 0) {
        LOG(L_ERR, "handle_subscription(): Error while extracting presentity UID\n");
        goto error;
    }

    lock_pdomain(d);

    if (find_presentity_uid(d, &uid, &p) > 0) {
        if (create_presentity_only(_m, d, &p_uri, &uid, &p) < 0) {
            LOG(L_ERR, "handle_publish can't create presentity\n");
            goto error2;
        }
    }
    str_free_content(&uid);

    LOG(L_DBG, "handle_publish - publishing status\n");

    /* update presentity event state */
    if (p) publish_presentity(_m, d, p);

    unlock_pdomain(d);

    if (send_reply(_m) < 0) {
    DBG("error on sending Ok back\n");
    return -1;
```

```
    }

    LOG(L_DBG, "handle_publish finished\n");
    return 1;

error2:
    unlock_pdomain(d);
    str_free_content(&uid);
error:
    send_reply(_m);
    return 0;
}
```

F.6. Notify.c

```
.
.
.
int send_notify(struct presentity* _p, struct watcher* _w)
{
    LOG(L_DBG, "\n\n send_notify: I was called with event package %d \n\n", _w->event_package);
    int rc = 0;
    //DBG("lalalalalalalalalla inside send_notify");
    //LOG(L_DBG, "notifying %.*s _p->flags=%x _w->event_package=%d _w-
>preferred_mimetype=%d _w->status=%d\n",
      //  _w->uri.len, _w->uri.s, _p->flags, _w->event_package, _w->preferred_mimetype, _w-
>status);

    if ((_w->status == WS_PENDING) ||
            (_w->status == WS_PENDING_TERMINATED) ||
            (_w->status == WS_REJECTED)) {
        notify_unauthorized_watcher(_p, _w);
        return 0;
    }

    switch (_w->event_package) {
        case EVENT_PRESENCE:
            rc = send_presence_notify(_p, _w);
            if (rc) LOG(L_ERR, "send_presence_notify returned %d\n", rc);
            break;

        /* Mohammad's code */

        case EVENT_LOCATION:
            rc = send_presence_notify(_p, _w);
            if (rc) LOG(L_ERR, "send_location-notify returned %d\n", rc);
            break;

        case EVENT_ROOMA:
            rc = send_presence_notify(_p, _w);
            if (rc) LOG(L_ERR, "send_ROOMA-notify returned %d\n", rc);
```

```
            break;

        case EVENT_ROOMB:
            rc = send_presence_notify(_p, _w);
            if (rc) LOG(L_ERR, "send_ROOMB-notify returned %d\n", rc);
            break;

        case EVENT_ROOMC:
            rc = send_presence_notify(_p, _w);
            if (rc) LOG(L_ERR, "send_ROOMC-notify returned %d\n", rc);
            break;

        /* Mohammad's code */

        case EVENT_PRESENCE_WINFO:
            rc = send_winfo_notify(_p, _w);
            if (rc < 0) LOG(L_ERR, "send_winfo_notify returned %d\n", rc);
            break;
        default: LOG(L_ERR, "sending notify for unknow package\n");
    }

/* FIXME: will be removed */
#if 0
#ifdef HAVE_XCAP_CHANGE_NOTIFY
    if ((_p->flags & PFLAG_XCAP_CHANGED)
        && (_w->event_package == EVENT_XCAP_CHANGE)) {
        switch(_w->preferred_mimetype) {
#ifdef DOC_XCAP_CHANGE
        case DOC_XCAP_CHANGE:
#endif
        default:
            rc = send_xcap_change_notify(_p, _w);
            if (rc) LOG(L_ERR, "send_xcap_change_notify returned %d\n", rc);
        }
    }
#endif /* HAVE_XCAP_CHANGE_NOTIFY */
#ifdef PFLAG_LOCATION_CHANGED
    if ((_p->flags & PFLAG_LOCATION_CHANGED)
        && (_w->event_package == EVENT_LOCATION)) {
        switch(_w->preferred_mimetype) {
        case DOC_LOCATION:
            rc = send_location_notify(_p, _w);
            if (rc) LOG(L_ERR, "send_location_notify returned %d\n", rc);
            break;
        default:
          rc = -1;
          ;
        }
    }
#endif /* PFLAG_LOCATION_CHANGED */
#endif
```

```
    return rc;
}
```

F.7. Pidf.c

```
#include <presence/pidf.h>
#include <cds/dstring.h>
#include <cds/memory.h>
#include <cds/logger.h>
#include <cds/list.h>
#include <presence/xml_utils.h>
#include <string.h>

#include "../modules/pa/presentity.h"

//char* description2, room2, floor2, latitude2, longtitude2, height2;


/* ------------------------------ PIDF document creation ------------------------------ */

static void doc_add_tuple_note(dstring_t *buf, presence_note_t *n)
{
    DEBUG_LOG("doc_add_tuple_note()\n");

    dstr_append_zt(buf, "\t\t<note");
    if (n->lang.len > 0) {
        dstr_append_zt(buf, " lang=\"");
        dstr_append_str(buf, &n->lang);
        dstr_append_zt(buf, "\"");
    }
    dstr_append_zt(buf, ">");
    dstr_append_str(buf, &n->value);
    dstr_append_zt(buf, "</note>\r\n");
}


static void doc_add_tuple(dstring_t *buf, presentity_info_t *p, presence_tuple_info_t *t)
{
    presence_note_t *n;
    char tmp[32];

    DEBUG_LOG("doc_add_tuple()\n");

    dstr_append_zt(buf, "\t<tuple id=\"");
    dstr_append_str(buf, &t->id);
    dstr_append_zt(buf, "\">\r\n");

    /*Mohammad For Location Event */
    if (t->status == presence_tuple_open)
    {
```

```
LOG(L_DBG, "\n\n befoere adding to description: [ %s ]\n\n", t->description.s);
LOG(L_DBG, "\n\n befoere adding to contact: [ %s ]\n\n", t->contact.s);
if (t->description.s==NULL)
{
dstr_append_zt(buf, "\t\t<status><basic>open</basic></status>\r\n");
}

else
{
   dstr_append_zt(buf, "\t\t<status><basic>open</basic>\r\n");

   dstr_append_zt(buf, "\t\t<location>\r\n");

   dstr_append_zt(buf, "\t\t<description>");
   dstr_append_str(buf, &t->description);
   dstr_append_zt(buf, "</description>\r\n");

   str_clear(&t->description);

   dstr_append_zt(buf, "\t\t<room>");
   dstr_append_str(buf, &t->room);
   dstr_append_zt(buf, "</room>\r\n");
   str_clear(&t->room);

   dstr_append_zt(buf, "\t\t<floor>");
   dstr_append_str(buf, &t->floor);
   dstr_append_zt(buf, "</floor>\r\n");
   str_clear(&t->floor);

   dstr_append_zt(buf, "\t\t<coordinates>\r\n");

   dstr_append_zt(buf, "\t\t<latitude>");
   dstr_append_str(buf, &t->latitude);
   dstr_append_zt(buf, "</latitude>\r\n");
   str_clear(&t->latitude);

   dstr_append_zt(buf, "\t\t<longtitude>");
   dstr_append_str(buf, &t->longtitude);
   dstr_append_zt(buf, "</longtitude>\r\n");
   str_clear(&t->longtitude);

   dstr_append_zt(buf, "\t\t<height>");
   dstr_append_str(buf, &t->height);
   dstr_append_zt(buf, "</height>\r\n");
   str_clear(&t->height);

   dstr_append_zt(buf, "\t\t</coordinates>\r\n");
   dstr_append_zt(buf, "\t\t</location>\r\n");
   dstr_append_zt(buf, "\t\t</status>\r\n");
}
```

```
    }
    /*Mohammad For Location Event */


    else dstr_append_zt(buf, "\t\t<status><basic>closed</basic></status>\r\n");

    dstr_append_zt(buf, "\t\t<contact priority=\"");
    sprintf(tmp, "%1.2f", t->priority);
    dstr_append_zt(buf, tmp);
    dstr_append_zt(buf, "\">");
    dstr_append_str(buf, &t->contact);
    dstr_append_zt(buf, "</contact>\r\n");

    n = t->first_note;
    while (n) {
        doc_add_tuple_note(buf, n);
        n = n->next;
    }

    dstr_append_zt(buf, "\t</tuple>\r\n");
}


static void doc_add_empty_tuple(dstring_t *buf)
{
    /* "empty" tuple is needed in PIDF by Microsoft Windows Messenger v. 5.1 and linphone 1.2) */
    DEBUG_LOG("doc_add_empty_tuple()\n");

    dstr_append_zt(buf, "\t<tuple id=\"none\">\r\n");
    dstr_append_zt(buf, "\t\t<status><basic>closed</basic></status>\r\n");

    dstr_append_zt(buf, "\t</tuple>\r\n");
}

static void doc_add_note(dstring_t *buf, presentity_info_t *p, presence_note_t *n)
{
    DEBUG_LOG("doc_add_note()\n");

    dstr_append_zt(buf, "\t<note");
    if (n->lang.len > 0) {
        dstr_append_zt(buf, " lang=\"");
        dstr_append_str(buf, &n->lang);
        dstr_append_zt(buf, "\"");
    }
    dstr_append_zt(buf, ">");
    dstr_append_str(buf, &n->value);
    dstr_append_zt(buf, "</note>\r\n");
}

static void doc_add_person(dstring_t *buf, presentity_info_t *p, person_t *ps)
{
```

```
    dstr_append_str(buf, &ps->person_element);
    dstr_append_zt(buf, "\r\n");
}

static void dstr_put_pres_uri(dstring_t *buf, str_t *uri)
{
    char *c;
    int len = 0;

    if (!uri) return;

    c = str_strchr(uri, ':');
    if (c) {
        len = uri->len - (c - uri->s) - 1;
        if (len > 0) c++;
    }
    else {
        c = uri->s;
        len = uri->len;
    }
    if (len > 0) {
        dstr_append_zt(buf, "pres:");
        dstr_append(buf, c, len);
    }
}

static void doc_add_presentity(dstring_t *buf, presentity_info_t *p, int use_cpim_pidf_ns)
{
    presence_tuple_info_t *t;
    presence_note_t *n;
    person_t *ps;

    //LOG(L_DBG, "\n here inside doc add presentity contact: [ %s ] \n", t-> contact.s);
    //LOG(L_DBG, "\n here inside doc add presentity description: [ %s ], room: [ %s ], floor: [ %s ],
latitude: [ %s ],        longtitude: [ %s ], height: [ %s ]  \n", t->description.s, t-> room.s , t->floor.s, t-
>latitude.s , t->longtitude.s , t->height.s);

    //it is ok up to here !!!!!!!!!!!!!!!!!!!!!!


    DEBUG_LOG("doc_add_presentity()\n");
    if (use_cpim_pidf_ns)
        dstr_append_zt(buf, "<presence xmlns=\"urn:ietf:params:xml:ns:cpim-pidf\" entity=\"");
    else
        dstr_append_zt(buf, "<presence xmlns=\"urn:ietf:params:xml:ns:pidf\" entity=\"");
    /* !!! there SHOULD be pres URI of presentity !!! */
    dstr_put_pres_uri(buf, &p->presentity);
    /* dstr_append_str(buf, &p->presentity); */ /* only for test !!! */
    dstr_append_zt(buf, "\">\r\n");

    DEBUG_LOG("adding tuples\n");
```

```
    t = p->first_tuple;
    if (!t) doc_add_empty_tuple(buf); /* correction for some strange clients :-) */
    while (t) {
        //Mohammad codes
        doc_add_tuple(buf, p, t);


        t = t->next;
    }

    DEBUG_LOG("adding notes\n");
    n = p->first_note;
    while (n) {
        doc_add_note(buf, p, n);
        n = n->next;
    }

    DEBUG_LOG("adding persons\n");
    ps = p->first_person;
    while (ps) {
        doc_add_person(buf, p, ps);
        ps = ps->next;
    }

    dstr_append_zt(buf, "</presence>\r\n");
}

int create_pidf_document_ex(presentity_info_t *p, str_t *dst, str_t *dst_content_type, int
use_cpim_pidf_ns)
{
    dstring_t buf;
    int err;

    if (!dst) return -1;

    str_clear(dst);
    if (dst_content_type) str_clear(dst_content_type);

    if (!p) return -1;

    if (dst_content_type) {
        if (use_cpim_pidf_ns)
            err = str_dup_zt(dst_content_type, "application/cpim-pidf+xml");
        else
            err = str_dup_zt(dst_content_type, "application/pidf+xml;charset=\"UTF-8\"");
        if (err < 0) return -1;
    }

/*    if (!p->first_tuple) return 0;*/    /* no tuples => nothing to say */

    dstr_init(&buf, 2048);
```

```
    dstr_append_zt(&buf, "<?xml version=\" 1.0\" encoding=\"UTF-8\"?>\r\n");
    doc_add_presentity(&buf, p, use_cpim_pidf_ns);

    err = dstr_get_str(&buf, dst);
    dstr_destroy(&buf);

    if (err != 0) {
        str_free_content(dst);
        if (dst_content_type) str_free_content(dst_content_type);
    }

    return err;
}

int create_pidf_document(presentity_info_t *p, str_t *dst, str_t *dst_content_type)
{
    return create_pidf_document_ex(p, dst, dst_content_type, 0);
}

/* ------------------------------ PIDF document parsing ------------------------------ */

static char *pidf_ns = "urn:ietf:params:xml:ns:pidf";
//static char *Mohammad_ns = "urn:ietf:params:xml:ns:pidf
xmlns:location=\"http://it.kth.se/~moze/schemas/mohammad.xsd\"";

/* static char *rpid_ns = "urn:ietf:params:xml:ns:pidf:rpid"; */
static char *data_model_ns = "urn:ietf:params:xml:ns:pidf:data-model";

static int read_note(xmlNode *node, presence_note_t **dst)
{
    const char *note = NULL;
    const char *lang = NULL;

    note = get_node_value(node);
    lang = get_attr_value(find_attr(node->properties, "lang"));

    *dst = create_presence_note_zt(note, lang);
    if (!dst) return -1;

    return 0;
}


static int read_tuple(xmlNode *tuple, presence_tuple_info_t **dst, int ignore_ns)
{
    str_t contact, id, description, room, floor, latitude, longtitude, height;
    presence_tuple_status_t status;
    xmlNode *n;
    double priority = 0;
    const char *s;
```

```
    int res = 0;
    presence_note_t *note;
    char *ns = ignore_ns ? NULL: pidf_ns;

    *dst = NULL;

/*    const char *description= NULL, *room=NULL, *floor=NULL, *latitude=NULL,
*longtitude=NULL, *height=NULL;
    description = (char *)malloc(sizeof(char) * 20);
    room = (char *)malloc(sizeof(char) * 20);
    floor = (char *)malloc(sizeof(char) * 20);
    latitude = (char *)malloc(sizeof(char) * 20);
    longtitude = (char *)malloc(sizeof(char) * 20);
    height = (char *)malloc(sizeof(char) * 20);  */

    str_clear(&description);
    str_clear(&room);
    str_clear(&floor);
    str_clear(&latitude);
    str_clear(&longtitude);
    str_clear(&height);


    DEBUG_LOG("read_tuple()\n");
    /* process contact (only one node) */
    n = find_node(tuple, "contact", ns);
    if (!n) {
        /* ERROR_LOG("contact not found\n"); */
        str_clear(&contact);
        /* return -1; */
    }
    else {
        s = get_attr_value(find_attr(n->properties, "priority"));
        if (s) priority = atof(s);
        s = get_node_value(n);
        contact.s = (char *)s;
        if (s) contact.len = strlen(s);
        else contact.len = 0;
        if (contact.len < 1) {
            ERROR_LOG("empty contact using default\n");
            /* return -1; */
        }
    }


    /* process status (only one node) */
    n = find_node(tuple, "status", ns);
    if (!n) {
        ERROR_LOG("status not found\n");
        return -1;
    }
```

```
/*Mohammad code for location tags */
n = find_node(n, "location", ns);
if (!n) {
   ERROR_LOG("\n\nlocation tag not found \n\n");
str_clear(&description);
str_clear(&room);
str_clear(&floor);
str_clear(&latitude);
str_clear(&longtitude);
str_clear(&height);
   /* return -1; */
}

else
{
   n = find_node(n, "description", ns);
   if (!n) {
      ERROR_LOG("\n\nDescription tag not found \n\n");
      /* return -1; */
      str_clear(&description);
   }
   else
   {
       s = get_node_value(n);
      description.s = (char *)s;
      if (s) description.len = strlen(s);
      else description.len = 0;
      LOG(L_DBG, "\n\n description length: [ %d ]\n\n", description.len);


   }

   n = find_node(tuple, "status", ns);
   n = find_node(n, "location", ns);
   n = find_node(n, "room", ns);
   if (!n) {
      ERROR_LOG("\n\nroom tag not found \n\n");
      /* return -1; */
      str_clear(&room);
   }
   else
   {
      s = get_node_value(n);
      room.s = (char *)s;
      if (s) room.len = strlen(s);
      else room.len = 0;

      //LOG(L_DBG, "\n room: [ %s ] \n", room);
   }
```

```
n = find_node(tuple, "status", ns);
n = find_node(n, "location", ns);
n = find_node(n, "floor", ns);
if (!n) {
   ERROR_LOG("\n\nfloor tag not found \n\n");
   /* return -1; */
   str_clear(&floor);
}
else
{
   s = get_node_value(n);
   floor.s = (char *)s;
   if (s) floor.len = strlen(s);
   else floor.len = 0;

   //LOG(L_DBG, "\n floor: [ %s ] \n", floor);
}


n = find_node(tuple, "status", ns);
n = find_node(n, "location", ns);
n = find_node(n, "coordinates", ns);
if (!n) {
   ERROR_LOG("\n\ncoordinates tag not found \n\n");
   /* return -1; */
   str_clear(&latitude);
   str_clear(&longtitude);
   str_clear(&height);
}
else
{
   n = find_node(n, "latitude", ns);
   s = get_node_value(n);
   latitude.s = (char *)s;
   if (s) latitude.len = strlen(s);
   else latitude.len = 0;

   //LOG(L_DBG, "\n latitude: [ %s ] \n", latitude);

   n = find_node(tuple, "status", ns);
   n = find_node(n, "location", ns);
   n = find_node(n, "coordinates", ns);
   n = find_node(n, "longtitude", ns);
   s = get_node_value(n);
   longtitude.s = (char *)s;
   if (s) longtitude.len = strlen(s);
   else longtitude.len = 0;

   //LOG(L_DBG, "\n longtitude: [ %s ] \n", longtitude);
```

```
        n = find_node(tuple, "status", ns);
        n = find_node(n, "location", ns);
        n = find_node(n, "coordinates", ns);
        n = find_node(n, "height", ns);
        s = get_node_value(n);
        height.s = (char *)s;
        if (s) height.len = strlen(s);
        else height.len = 0;

        LOG(L_DBG, "\n description: [ %s ], room: [ %s ], floor: [ %s ], latitude: [ %s ], longtitude:
[ %s ], height: [ %s ]  \n", description.s, room.s , floor.s, latitude.s , longtitude.s , height.s);
      }


   }
   /*Mohammad code */


   n = find_node(tuple, "status", ns);
   n = find_node(n, "basic", ns);
   if (!n) {
      ERROR_LOG("basic status not found - using \'closed\'\n");
      /* return -1; */
      s = "closed";
   }
   else s = get_node_value(n);
   if (!s) {
      ERROR_LOG("basic status without value\n");
      return -1;
   }

   /* translate status */
   status = presence_tuple_closed; /* default value */
   if (strcmp(s, "open") == 0) status = presence_tuple_open;
   if (strcmp(s, "closed") == 0) status = presence_tuple_closed;
   /* FIXME: handle not standardized variants too (add note to basic status) */



   /* get ID from tuple node attribute? */
   id.s = (char *)get_attr_value(find_attr(tuple->properties, "id"));
   if (id.s) id.len = strlen(id.s);
   else id.len = 0;

   *dst = create_tuple_info(&contact, &id, status, &description, &room, &floor, &latitude,
&longtitude, &height);
   if (!(*dst)) return -1;

   (*dst)->priority = priority;
```

```
    /* handle notes */
    n = tuple->children;
    while (n) {
        if (n->type == XML_ELEMENT_NODE) {
            if (cmp_node(n, "note", ns) >= 0) {
                res = read_note(n, &note);
                if ((res == 0) && note) {
                    DOUBLE_LINKED_LIST_ADD((*dst)->first_note,
                            (*dst)->last_note, note);
                }
                else break;
            }
        }
        n = n->next;
    }
    //LOG(L_DBG, "\n description2: [ %s ] \n", description2);

    return res;
}

static int get_whole_node_content(xmlNode *n, str_t *dst, xmlDocPtr doc)
{
    int res = 0;

    str_clear(dst);
    if (n) {
        n = xmlCopyNode(n, 1); /* this inserts namespaces into element correctly */
        if (!n) {
            ERROR_LOG("can't duplicate XML node\n");
            return -1;
        }
    }
    if (n) {
        xmlBufferPtr buf;
        buf = xmlBufferCreate();
        if (buf == NULL) {
            ERROR_LOG("Error creating the xml buffer\n");
            return -1;
        }
        if (xmlNodeDump(buf, doc, n, 0, 0) < 0) res = -1;
        if ((res == 0) && (buf->use > 0)) {
            str_t s;
            s.s = (char *)buf->content;
            s.len = buf->use;
            res = str_dup(dst, &s);
        }
        xmlBufferFree(buf);
        xmlFreeNode(n); /* was duplicated due to namespaces! */
    }
    return res;
```

```
}

static int read_person(xmlNode *person, person_t **dst, xmlDocPtr doc)
{
    person_t *p;
    /* xmlNode *n; */

    if (!dst) return -1;
    *dst = NULL;

    p = (person_t*)cds_malloc(sizeof(person_t));
    if (!p) return -1;

    memset(p, 0, sizeof(*p));
    *dst = p;

    TRACE_LOG("reading person ()\n");

    if (str_dup_zt(&p->id, get_attr_value(find_attr(person->properties, "id"))) < 0) {
        cds_free(p);
        *dst = NULL;
        return -1;
    }

    /* try to find mood */
/*    n = find_node(person, "mood", rpid_ns);
    if (n) get_whole_node_content(n, &p->mood, doc);
*/
    /* try to find activities */
/*    n = find_node(person, "activities", rpid_ns);
    if (n) get_whole_node_content(n, &p->activities, doc);
*/
    /* do not care about internals of person - take whole element ! */
    if (get_whole_node_content(person, &p->person_element, doc) != 0) {
        str_free_content(&p->id);
        cds_free(p);
        *dst = NULL;
        return -1;
    }

    return 0;
}

static int read_presentity(xmlNode *root, presentity_info_t **dst, int ignore_ns, xmlDocPtr doc)
{
    xmlNode *n;
    str_t entity;
    presence_tuple_info_t *t;

    //presence_tuple_t *pr;
```

```
    presence_note_t *note;
    int res = 0;
    char *ns = ignore_ns ? NULL: pidf_ns;
    person_t *p, *lastp;

    DEBUG_LOG("read_presentity(ns=%s)\n", ns ? ns : "");
    if (cmp_node(root, "presence", ns) < 0) {
        ERROR_LOG("document is not presence \n");
        return -1;
    }

    entity = zt2str((char*)get_attr_value(find_attr(root->properties, "entity")));
    *dst = create_presentity_info(&entity);
    if (!(*dst)) return -1; /* memory */

    lastp = NULL;
    n = root->children;
    while (n) {
        if (n->type == XML_ELEMENT_NODE) {
            if (cmp_node(n, "tuple", ns) >= 0) {
                res = read_tuple(n, &t, ignore_ns);
                if ((res == 0) && t) add_tuple_info(*dst, t);
                else break;
            }
            if (cmp_node(n, "note", ns) >= 0) {
                res = read_note(n, &note);
                if ((res == 0) && note) {
                    DOUBLE_LINKED_LIST_ADD((*dst)->first_note,
                            (*dst)->last_note, note);
                }
                else break;
            }

            /* RPID extensions */
            if (cmp_node(n, "person", data_model_ns) >= 0) {
                p = NULL;
                res = read_person(n, &p, doc);
                if ((res == 0) && p)
                    LINKED_LIST_ADD((*dst)->first_person, lastp, p);
                /*if (res != 0) break; ignore errors there */
            }

        }
        n = n->next;
    }

    return res;
}

/* ignore ns added for cpim-pidf+xml, draft version 07 (differs only in ns) */
int parse_pidf_document_ex(presentity_info_t **dst, const char *data, int data_len, int ignore_ns)
```

```
{
    int res = 0;
    xmlDocPtr doc;

    if (!dst) return -1;
    if ((!data) || (data_len < 1)) return -2;

    *dst = NULL;
    doc = xmlReadMemory(data, data_len, NULL, NULL, xml_parser_flags);
    if (doc == NULL) {
        ERROR_LOG("can't parse document\n");
        return -1;
    }

    res = read_presentity(xmlDocGetRootElement(doc), dst, ignore_ns, doc);
    if (res != 0) {
        /* may be set => must be freed */
        if (*dst) free_presentity_info(*dst);
        *dst = NULL;
    }

    xmlFreeDoc(doc);
    return res;
}

/* libxml2 must be initialized before calling this function ! */
int parse_pidf_document(presentity_info_t **dst, const char *data, int data_len)
{
    return parse_pidf_document_ex(dst, data, data_len, 0);
}

/* ---------- CPIM_PIDF document creation/parsing ---------- */

int parse_cpim_pidf_document(presentity_info_t **dst, const char *data, int data_len)
{
    return parse_pidf_document_ex(dst, data, data_len, 1);
}

int create_cpim_pidf_document(presentity_info_t *p, str_t *dst, str_t *dst_content_type)
{
    return create_pidf_document_ex(p, dst, dst_content_type, 1);
}
```

F.8. Tuple_notes.c

.
.
.

```
int db_remove_tuple_notes(presentity_t *p, presence_tuple_t *t)
{
    db_key_t keys[] = { "presid", "tupleid" };
```

```
    db_op_t ops[] = { OP_EQ, OP_EQ };
    db_val_t k_vals[] = {
        { DB_INT, 0, { .int_val = p->presid } },
        { DB_STR, 0, { .str_val = t->id } }
    };

    if (!use_db) return 0;

    if (pa_dbf.use_table(pa_db, tuple_notes_table) < 0) {
        LOG(L_ERR, "db_remove_tuple_notes: Error in use_table\n");
        return -1;
    }

    if (pa_dbf.delete(pa_db, keys, ops, k_vals, 2) < 0) {
        LOG(L_ERR, "db_remove_tuple_notes: Can't delete record\n");
        return -1;
    }

    //Mohammad code for removing event table//

    db_key_t keys2[] = { "presid", "tupleid" };
    db_op_t ops2[] = { OP_EQ, OP_EQ };
    db_val_t k_vals2[] = {
        { DB_INT, 0, { .int_val = p->presid } },
        { DB_STR, 0, { .str_val = t->published_id } }
    };

    if (pa_dbf.use_table(pa_db, presentity_event_table) < 0) {
        LOG(L_ERR, "db_remove_event: Error in use_table\n");
        return -1;
    }

    LOG(L_ERR, "\n\n before removing from event table \n\n");

    if (pa_dbf.delete(pa_db, keys2, ops2, k_vals2, 2) < 0) {
        LOG(L_ERR, "db_remove_event: Can't delete record\n");
        return -1;
    }
    //Mohammad code for removing event table//


    return 0;
}
.
.
.
```

F.9. Pa_mod.c

```
#include <signal.h>

#include "../../db/db.h"
#include "../../sr_module.h"
#include "../../error.h"
#include "subscribe.h"
#include "publish.h"
#include " dlist.h"
#include "location.h"
#include "pa_mod.h"
#include "watcher.h"
#include "rpc.h"
#include "qsa_interface.h"

#include <cds/logger.h>
#include <cds/cds.h>
#include <presence/qsa.h>

#include "status_query.h"
#include "offline_winfo.h"
#include "message.h"

MODULE_VERSION

static int pa_mod_init(void);  /* Module initialization function */
static int pa_child_init(int _rank);  /* Module child init function */
static void pa_destroy(void);  /* Module destroy function */
static int subscribe_fixup(void** param, int param_no); /* domain name -> domain pointer */
static void timer(unsigned int ticks, void* param); /* Delete timer for all domains */

int default_expires = 3600;  /* Default expires value if not present in the message (for SUBSCRIBE
and PUBLISH) */
int max_subscription_expiration = 3600;  /* max expires value for subscribe */
int max_publish_expiration = 3600;  /* max expires value for subscribe */
int timer_interval = 10;     /* Expiration timer interval in seconds */
double default_priority = 0.0; /* Default priority of presence tuple */
static int default_priority_percentage = 0; /* expressed as percentage because config file grammar
does not support floats */
int watcherinfo_notify = 1; /* send watcherinfo notifications */

/** TM bind */
struct tm_binds tmb;

dlg_func_t dlg_func;


/** database */
db_con_t* pa_db = NULL; /* Database connection handle */
db_func_t pa_dbf;
```

```
int use_db = 1;
str db_url = STR_NULL;
int use_place_table = 0;
#ifdef HAVE_LOCATION_PACKAGE
str pa_domain = STR_NULL;
#endif /* HAVE_LOCATION_PACKAGE */
char *presentity_table = "presentity";
char *presentity_contact_table = "presentity_contact";
char *presentity_event_table = "presentity_event";
char *presentity_notes_table = "presentity_notes";

char *presentity_locations_table = "presentity_locations";

char *person_elements_table = "presentity_persons";
char *tuple_notes_table = "tuple_notes";
char *watcherinfo_table = "watcherinfo";
char *place_table = "place";
char *offline_winfo_table = "offline_winfo";

/* authorization parameters */
char *auth_type_str = NULL; /* type of authorization */
char *auth_xcap_root = NULL;    /* must be set if xcap authorization */
char *winfo_auth_type_str = "implicit"; /* type of authorization */
char *winfo_auth_xcap_root = NULL;    /* must be set if xcap authorization */
auth_params_t pa_auth_params;    /* structure filled according to parameters */
auth_params_t winfo_auth_params;    /* structure for watcherinfo filled according to parameters */

int use_bsearch = 0;
int use_location_package = 0;

/* use callbacks to usrloc/??? - if 0 only pusblished information is used */
int use_callbacks = 1;
int use_offline_winfo = 0;
int offline_winfo_timer_interval = 3600;

/*
 * Exported functions
 */
static cmd_export_t cmds[]={
   {"handle_subscription",   handle_subscription,   1, subscribe_fixup, REQUEST_ROUTE |
FAILURE_ROUTE},
   {"handle_publish",        handle_publish,        1, subscribe_fixup, REQUEST_ROUTE |
FAILURE_ROUTE},

   {"target_online",         target_online,         1, subscribe_fixup, REQUEST_ROUTE |
FAILURE_ROUTE},
   {"store_winfo",           store_offline_winfo,   1, 0, REQUEST_ROUTE | FAILURE_ROUTE},
   {"dump_stored_winfo",     dump_offline_winfo,    2, subscribe_fixup, REQUEST_ROUTE |
FAILURE_ROUTE},

   /* TODO: move into XCAP module */
```

```
    {"authorize_message",     authorize_message,    1, 0, REQUEST_ROUTE | FAILURE_ROUTE},

    /* FIXME: are these functions used to something by somebody */
/*
 *
    {"pua_exists",           pua_exists,           1, subscribe_fixup, REQUEST_ROUTE              },
    {"pa_handle_registration", pa_handle_registration,   1, subscribe_fixup, REQUEST_ROUTE },
    {"existing_subscription", existing_subscription, 1, subscribe_fixup, REQUEST_ROUTE             },
     {"mangle_pidf",          mangle_pidf,          0, NULL, REQUEST_ROUTE | FAILURE_ROUTE},
    {"mangle_message_cpim",   mangle_message_cpim,   0, NULL,          REQUEST_ROUTE |
FAILURE_ROUTE},*/

    {0, 0, 0, 0, 0}
};


/*
 * Exported parameters
 */
static param_export_t params[]={
    {"default_expires",      PARAM_INT,    &default_expires      },
    {"max_subscription_expiration", PARAM_INT, &max_subscription_expiration },
    {"max_publish_expiration", PARAM_INT, &max_publish_expiration },

    {"auth",                 PARAM_STRING, &auth_type_str }, /* type of authorization: none, implicit,
xcap, ... */
    {"auth_xcap_root",       PARAM_STRING, &auth_xcap_root }, /* xcap root settings - must be set
for xcap auth */
    {"winfo_auth",           PARAM_STRING, &winfo_auth_type_str }, /* type of authorization: none,
implicit, xcap, ... */
    {"winfo_auth_xcap_root", PARAM_STRING, &winfo_auth_xcap_root }, /* xcap root settings -
must be set for xcap auth */

    {"use_db",               PARAM_INT,    &use_db               },
    {"use_callbacks", PARAM_INT, &use_callbacks  }, /* use callbacks to usrloc/jabber ? */
    {"accept_internal_subscriptions", PARAM_INT, &accept_internal_subscriptions },
    {"watcherinfo_notify",   PARAM_INT, &watcherinfo_notify    }, /* accept winfo subscriptions ? */

    {"use_offline_winfo", PARAM_INT, &use_offline_winfo  }, /* use DB for offline winfo */
    {"offline_winfo_expiration", PARAM_INT, &offline_winfo_expiration }, /* how long hold
information in DB */
    {"offline_winfo_timer", PARAM_INT, &offline_winfo_timer_interval }, /* basic ticks of "offline
winfo" timer */

    {"db_url",               PARAM_STR,    &db_url               },

    /* undocumented still (TODO) */
    {"presentity_table",     PARAM_STRING, &presentity_table     },
    {"presentity_contact_table", PARAM_STRING, &presentity_contact_table     },
    {"presentity_event_table", PARAM_STRING, &presentity_event_table     },
    {"watcherinfo_table",    PARAM_STRING, &watcherinfo_table    },
```

```
    {"place_table",          PARAM_STRING, &place_table          },
    {"default_priority_percentage", PARAM_INT,    &default_priority_percentage  },
    {"timer_interval",       PARAM_INT,    &timer_interval       },
    {"use_place_table",      PARAM_INT,    &use_place_table      },
    {"use_bsearch",          PARAM_INT,    &use_bsearch          },
    {"use_location_package", PARAM_INT,    &use_location_package },
#ifdef HAVE_LOCATION_PACKAGE
    {"pa_domain",            PARAM_STR,    &pa_domain            },
#endif /* HAVE_LOCATION_PACKAGE */
    {"offline_winfo_table", PARAM_STRING, &offline_winfo_table }, /* table with offline winfo */

    {0, 0, 0}
};


struct module_exports exports = {
    "pa",
    cmds,          /* Exported functions */
    pa_rpc_methods, /* RPC methods */
    params,        /* Exported parameters */
    pa_mod_init,    /* module initialization function */
    0,             /* response function*/
    pa_destroy,     /* destroy function */
    0,             /* oncancel function */
    pa_child_init   /* per-child init function */
};
.
.
.

My_create.sql
CREATE TABLE presentity_event (
    presid INT(10) UNSIGNED NOT NULL,
    event VARCHAR(128) NOT NULL,
    tupleid VARCHAR(64) NOT NULL,
    eventid INT(10) UNSIGNED AUTO_INCREMENT NOT NULL,
    UNIQUE KEY pc_idx1 (eventid)
);
```

F.10. presentity.h

```
#ifndef PRESENTITY_H
#define PRESENTITY_H

#include "../../str.h"
#include "../tm/dlg.h"
#include "watcher.h"
#include "hslot.h"
#include "pstate.h"
#include "trace.h"
```

```
#include <xcap/pres_rules.h>
#include <cds/msg_queue.h>
#include <presence/notifier.h>
#include <presence/pres_doc.h>

enum prescaps {
   PRESCAP_AUDIO = (1 << 0),
   PRESCAP_VIDEO = (1 << 1),
   PRESCAP_TEXT = (1 << 2),
   PRESCAP_APPLICATION = (1 << 3)
};
extern const char *prescap_names[];

#define TUPLE_STATUS_STR_LEN 128
#define TUPLE_LOCATION_LOC_LEN 128
#define TUPLE_LOCATION_SITE_LEN 32
#define TUPLE_LOCATION_FLOOR_LEN 32
#define TUPLE_LOCATION_ROOM_LEN 64
#define TUPLE_LOCATION_PACKET_LOSS_LEN 32
#define TUPLE_ID_STR_LEN (32)

typedef struct location {
   str   loc; /* human readable description of location */
   str   site;
   str   floor;
   str   room;
   str   packet_loss;
   double x;
   double y;
   double radius;
   char loc_buf[TUPLE_LOCATION_LOC_LEN];
   char site_buf[TUPLE_LOCATION_SITE_LEN];
   char floor_buf[TUPLE_LOCATION_FLOOR_LEN];
   char room_buf[TUPLE_LOCATION_ROOM_LEN];
   char packet_loss_buf[TUPLE_LOCATION_PACKET_LOSS_LEN];
} location_t;

typedef struct resource_list {
   str   uri;
   struct resource_list *next;
   struct resource_list *prev;
} resource_list_t;

typedef struct location_package {
   resource_list_t *users;
   resource_list_t *phones;
} location_package_t;

typedef struct presence_tuple {
   str id;
   str contact;
```

```
    str status;
    str event;    //Mohammad code for event
    str description;
    str room;
    str floor;
    str latitude;
    str longtitude;
    str height;
    enum prescaps prescaps;
    double priority;
    time_t expires;
    pstate_t state;
    location_t location;
    struct presence_tuple *next;
    struct presence_tuple *prev;
    char status_buf[TUPLE_STATUS_STR_LEN];
    char id_buf[TUPLE_ID_STR_LEN];
    int is_published;    /* 1 for published tuples - these are stored into DB */

    //presence_tuple_info_t data; by Mohammad

    str etag;    /* etag for published tuples */
    str published_id;    /* tuple id used for publish */
    presence_note_t *notes; /* notes for this tuple */

} presence_tuple_t;
.
.
.
```

F.11. presentity.c

```
.
.
.
static void process_watchers(presentity_t* _p, int *changed)
{
    watcher_t *next, *w, *prev;
    int presentity_changed;
    int notify;

    /* !!! "changed" is not initialized here it is only set if change
     * in presentity occurs */

    presentity_changed = _p->flags & (PFLAG_PRESENCE_CHANGED
            | PFLAG_PRESENCE_LISTS_CHANGED
            | PFLAG_XCAP_CHANGED
            | PFLAG_LOCATION_CHANGED);

    prev = NULL;
    w = _p->watchers;
```

```
   while (w) {
      /* changes status of expired watcher */
      if (w->expires <= act_time) {
         LOG(L_DBG, "Expired watcher %.*s\n", w-> uri.len, w->uri.s);
         w->expires = 0;
         set_watcher_terminated_status(w);
         _p->flags |= PFLAG_WATCHERINFO_CHANGED;
         w->flags |= WFLAG_SUBSCRIPTION_CHANGED;
         if (changed) *changed = 1;
      }

      /* send NOTIFY if needed */
      notify = 0;
      if ((w->flags & WFLAG_SUBSCRIPTION_CHANGED)) {
         notify = 1;
         if (changed) *changed = 1; /* ??? */
      }
      if (presentity_changed && is_watcher_authorized(w)) notify = 1;

      //Mohammad codes
      if (notify){
         // Mohammad Code for sending Notify to different events //
         //read the event from table

         //LOG(L_ERR, "\n\ntupleid inside processing watcher: [ %s ]\n\n",_p->tuples-
>published_id.s);


         db_con_t* pa_db = create_pa_db_connection();
         db_key_t keys[] = { "presid" };
         db_op_t ops[] = { OP_EQ };
         db_val_t k_vals[] = {
            { DB_INT, 0, { .int_val = _p->presid } }
         };

         //LOG(L_ERR, "\n\n ser hangs here \n\n");

         db_res_t *res = NULL;
         db_key_t result_cols[] = { "event" };

         if (pa_dbf.use_table(pa_db, presentity_event_table) < 0) {
            LOG(L_ERR, "db_read_event: Error in use_table\n");
            close_pa_db_connection(pa_db);
            }

         if (pa_dbf.query (pa_db, keys, ops, k_vals,
            result_cols, 1, 1, 0, &res) < 0) {
            LOG(L_ERR, "db_read_event: Error in reading table\n");
            close_pa_db_connection(pa_db);
            }
```

```
    LOG(L_ERR, "\n\n what is in table event [ %d ] \n\n",res->n);
  if (res && res->n > 0)
  {
    //struct watcher* watcherIterator = _p->watchers;

    if (strcmp(res->rows->values->val.string_val, "presence")==0 )
    {
      //LOG(L_ERR, "\n\n ser hangs here \n\n");
      LOG(L_ERR, "\n\n event query: [ %s ]\n\n", res->rows->values->val.string_val);

      //while(watcherIterator){

        if (w->event_package==EVENT_PRESENCE){
          LOG(L_ERR, "\n\nGoing to call send_notify with event id = [%d] \n\n", w-
>event_package);
           send_notify(_p, w);
          //_p->flags &= ~PFLAG_WATCHERINFO_CHANGED;

        }
        //watcherIterator = watcherIterator->next;
      //}

    }
    else if (strcmp(res->rows->values-> val.string_val, "location")==0 )
    {
      //while(watcherIterator){

        if (w->event_package==EVENT_LOCATION){
          LOG(L_ERR, "\n\nGoing to call send_notify with event id = [%d] \n\n", w-
>event_package);
           send_notify(_p, w);
          //_p->flags &= ~PFLAG_WATCHERINFO_CHANGED;
          LOG(L_ERR, "\n\n send Notify with location Event \n\n");

        }
        //watcherIterator = watcherIterator->next;
      //}

    }
    else if (strcmp(res->rows->values-> val.string_val, "roomA")==0 )
    {
      //while(watcherIterator){

        if (w->event_package==EVENT_ROOMA){
          LOG(L_ERR, "\n\nGoing to call send_notify with event id = [%d] \n\n", w-
>event_package);
           send_notify(_p, w);
          //_p->flags &= ~PFLAG_WATCHERINFO_CHANGED;
        }
      //   watcherIterator = watcherIterator->next;
```

```
            //}

        }
        else if (strcmp(res->rows->values->val.string_val, "roomB")==0 )
        {
            //while(watcherIterator){

                if (w->event_package==EVENT_ROOMB){
                    LOG(L_ERR, "\n\nGoing to call send_notify with event id = [%d] \n\n", w-
>event_package);
                     send_notify(_p, w);
                    //_p->flags &= ~PFLAG_WATCHERINFO_CHANGED;
                }
                //watcherIterator = watcherIterator->next;
            //}

        }
        else if (strcmp(res->rows->values-> val.string_val, "roomC")==0 )
        {
            //while(watcherIterator){

                if (w->event_package==EVENT_ROOMC){
                    LOG(L_ERR, "\n\nGoing to call send_notify with event id = [%d] \n\n", w-
>event_package);
                     send_notify(_p, w);
                    //_p->flags &= ~PFLAG_WATCHERINFO_CHANGED;
                }
                //watcherIterator = watcherIterator->next;
            //}

        }

        else
            ;
    }
    //if (res->n == 0)    send_notify(_p, w);   //send Notify when presentity expires
  }  //closing if(notify) here

    w->flags &= ~WFLAG_SUBSCRIPTION_CHANGED;

    if (is_watcher_terminated(w)) {
        next = w->next;
        if (prev) prev->next = next;
        else _p->watchers = next;
        if (use_db) db_remove_watcher(_p, w);
        free_watcher(w);
        w = next;
        _p->flags |= PFLAG_WATCHERINFO_CHANGED; /* terminated status could be set before
*/
        if (changed) *changed = 1;
    }
```

```
        else {
            prev = w;
            w = w->next;
        }
    }
}
.
.
.
```

Appendix G

M1:

Request-Line: SUBSCRIBE sip:ccsleft@130.237.15.238 SIP/2.0
Method: SUBSCRIBE
[Resent Packet: False]
Message Header
Via: SIP/2.0/UDP 130.237.15.238:5060;branch=z9hG4lfw8Wu0E9
Transport: UDP
Sent-by Address: 130.237.15.238
Sent-by port: 5060
Branch: z9hG4lfw8Wu0E9
To: <sip:ccsleft@130.237.15.238>
SIP to address: sip:ccsleft@130.237.15.238
From: <sip:Sub1@130.237.15.238>;tag=w8Wu
SIP from address: sip:Sub1@130.237.15.238
SIP tag: w8Wu
Call-ID: 728@130.237.15.238
CSeq: 764 SUBSCRIBE
Sequence Number: 764
Method: SUBSCRIBE
Max-Forwards: 70
Event: location
Accept: application/pidf+xml
Contact: <sip:Sub1@130.237.238.112>
Contact Binding: <sip:Sub1@130.237.238.112>
URI: <sip:Sub1@130.237.238.112>
SIP contact address: sip:Sub1@130.237.238.112
Expires: 5000
Content-Length: 0

M2:

Status-Line: SIP/2.0 200 OK
Status-Code: 200
[Resent Packet: False]
Message Header
Via: SIP/2.0/UDP
130.237.15.238:5060;branch=z9hG
4lfw8Wu0E9;received=130.237.238
.112
Transport: UDP
Sent-by Address: 130.237.15.238
Sent-by port: 5060
Branch: z9hG4lfw8Wu0E9

Received: 130.237.238.112
To:
<sip:ccsleft@130.237.15.238>;tag=d(
2a7583d137144-ed d5
SIP to address: sip:ccsleft@130.237.15.238
SIP tag: d6bf04da8d94fdfca282a7583d137144-edd5
From: <sip:Sub1@130.237.15.238>;tag=w8Wu
SIP from address: sip:Sub1@130.237.15.238
SIP tag: w8Wu
Call-ID: 728@130.237.15.238
CSeq: 764 SUBSCRIBE
Sequence Number: 764
Method: SUBSCRIBE
Expires: 600
Contact: <sip:130.237.15.238:5060>
Contact Binding: <sip:130.237.15.238:5060>
URI: <sip:130.237.15.238:5060>
SIP contact address: sip:130.237.15.238:5060
Server: Sip EXpress router (0.10.99-dev35-pa-4.1 (i386/linux))
Content-Length: 0

M3:
Request-Line: NOTIFY sip:Sub1@130.237.238.112 SIP/2.0
Method: NOTIFY
[Resent Packet: False]
Message Header
Via: SIP/2.0/UDP 130.237.15.238;branch=z9hG4bK46c1.b2e589e3.0
Transport: UDP
Sent-by Address: 130.237.15.238
Branch: z9hG4bK46c1.b2e589e3.0
To: <sip:Sub1@130.237.15.238>;tag=w8Wu
SIP to address: sip:Sub1@130.237.15.238
SIP tag: w8Wu
From:
<sip:ccsleft@130.237.15.238>;tag=
d6bf04da8d94fdfca282a7583d1371
44- edd5
SIP from address: sip:ccsleft@130.237.15.238
SIP tag: d6bf04da8d94fdfca282a7583d137144-edd5
CSeq: 1 NOTIFY
Sequence Number: 1
Method: NOTIFY
Call-ID: 728@130.237.15.238
Content-Length: 211
User-Agent: Sip EXpress router(0.10.99-dev35-pa-4.1 (i386/linux))

Event: location
Content-Type: application/pidf+xml;charset="UTF-8"
Contact: <sip:130.237.15.238:5060>
Contact Binding: <sip:130.237.15.238:5060>
URI: <sip:130.237.15.238:5060>
SIP contact address: sip:130.237.15.238:5060
Subscription-State: active;expires=600
Message body
eXtensible Markup Language
<?xml version="1.0"
encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf" entity="pres:ccsleft@130.237.15.238">
<tuple id="none">
<status>
<basic> closed</basic>
</status>
</tuple>
</presence>

M4:
Status-Line: SIP/2.0 200 OK
Status-Code: 200
[Resent Packet: False]
Message Header
Via: SIP/2.0/UDP 130.237.15.238;branch=z9hG4bK46c1.b2e589e3.0;received=1
Transport: UDP
Sent-by Address: 130.237.15.238
Branch: z9hG4bK46c1.b2e589e3.0
Received: 130.237.238.112
To: <sip:Sub1@130.237.15.238>;tag=w8Wu
SIP to address: sip:Sub1@130.237.15.238
SIP tag: w8Wu
From: <sip:ccsleft@130.237.15.238>;tag=d6bf04da8d94fdfca282a7583d137144-
SIP from address: sip:ccsleft@130.237.15.238
SIP tag:
d6bf04da8d94fdfca282a7583d1371
44-edd5
CSeq: 1 NOTIFY
Sequence Number: 1
Method: NOTIFY
Event: location
Message body
Content-Length: 0

M5:

Request-Line: PUBLISH sip:ccsleft@130.237.15.238 SIP/2.0
Method: PUBLISH
[Resent Packet: False]
Message Header
Via: SIP/2.0/UDP 130.237.238.87:5060;branch=z9hG4bKO8hr1ue
Transport: UDP
Sent-by Address: 130.237.238.87
Sent-by port: 5060
Branch: z9hG4bKO8hr1ue
To: <sip:ccsleft@130.237.15.238>
SIP to address: sip:ccsleft@130.237.15.238
From: <sip:ccsleft@130.237.15.238>;tag=n0o4
SIP from address: sip:ccsleft@130.237.15.238
SIP tag: n0o4
Call-ID: Ti5hPY17Fp11/6/07 4:25:31 PM@130.237.238.87
CSeq: 1 PUBLISH
Sequence Number: 1
Method: PUBLISH
Max-Forwards: 70
Expires: 10
Event: location
Content-Type: application/pidf+xml
Content-Length: 473
Message body
eXtensible Markup Language

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
xmlns:location="http://it.kth.se/~moze/schemas/mohammad.xsd"
entity="sip:ccsleft@130.237.15.238
">
<tuple id="GHLKOk">
<status>
  <basic> open</basic>
    <location>
      <description> Lab</description>

<room></room><floor></floor>
    <coordinates>

<latitude></latitude><longitude></l
ongitude>
   </coordinates>
   </location>
</status>
```

<note></note>
<contact
priority="0.8">130.237.238.87</con
tact>

</tuple>

</presence>

M6:

Status-Line: SIP/2.0 200 OK

Status-Code: 200

[Resent Packet: False]

Message Header

Via: SIP/2.0/UDP 130.237.238.87:5060;branch=z9hG4bKO8hr1ue

Transport: UDP

Sent-by Address: 130.237.238.87

Sent-by port: 5060

Branch: z9hG4bKO8hr1ue
To:
<sip:ccsleft@130.237.15.238>;tag=
d6bf04da8d94fdfca282a7583d1371
44-a1
SIP to address:
sip:ccsleft@130.237.15.238

SIP tag: d6bf04da8d94fdfca282a7583d137144-a1d0

From: <sip:ccsleft@130.237.15.238>;tag=n0o4

SIP from address: sip:ccsleft@130.237.15.238

SIP tag: n0o4

Call-ID: Ti5hPY17Fp11/6/07 4:25:31 PM@130.237.238.87

CSeq: 1 PUBLISH

Sequence Number: 1

Method: PUBLISH

Expires: 10

SIP-ETag: 0xb58f4f5cx6cf642cx47309446

Contact: <sip:130.237.15.238:5060>

Contact Binding: <sip:130.237.15.238:5060>

URI: <sip:130.237.15.238:5060>

SIP contact address: sip:130.237.15.238:5060

Content-Length: 0

M7:

Request-Line: NOTIFY sip:Sub1@130.237.238.112 SIP/2.0

Method: NOTIFY

[Resent Packet: False]

Message Header

Via: SIP/2.0/UDP 130.237.15.238;branch=z9hG4bK16c1.748a3562.0

Transport: UDP

Sent-by Address: 130.237.15.238
Branch: z9hG4bK16c1.748a3562.0
To: <sip:Sub1@130.237.15.238>;tag=w8Wu
SIP to address: sip:Sub1@130.237.15.238
SIP tag: w8Wu
From: <sip:ccsleft@130.237.15.238>;tag=d6bf04da8d94fdfca282a7583d137144-
SIP from address: sip:ccsleft@130.237.15.238
SIP tag: d6bf04da8d94fdfca282a7583d137144-edd5
CSeq: 2 NOTIFY
Sequence Number: 2
Method: NOTIFY
Call-ID: 728@130.237.15.238
Content-Length: 516
User-Agent: Sip EXpress router(0.10.99-dev35-pa-4.1 (i386/linux))
Event: location
Content-Type: application/pidf+xml;charset="UTF-8"
Contact: <sip:130.237.15.238:5060>
Contact Binding: <sip:130.237.15.238:5060>
URI: <sip:130.237.15.238:5060>
SIP contact address: sip:130.237.15.238:5060
Subscription-State: active;expires=589
Message body
eXtensible Markup Language

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf" entity="pres:ccsleft@130.237.15.238">
<tuple id="0xb58f5b14x14c5439bx47309446">
<status>
  <basic> Open</basic>
    <location>
      <description> Lab</description>
      <room></room>
      <floor></floor>
      <coordinates>
        <latitude></latitude>
        <longitude></longitude>
       <height></height>
      </coordinates>
    </location>
</status>
<contact priority="0.80"> 130.237.238.87</contact>
<note></note>
</tuple>
</presence>
```

M8:

Status-Line: SIP/2.0 200 OK

Status-Code: 200

Message Header

Via: SIP/2.0/UDP 130.237.15.238;branch=z9hG4bK16c1.748a3562.0;received=1

Transport: UDP

Sent-by Address: 130.237.15.238

Branch: z9hG4bK16c1.748a3562.0

Received: 130.237.238.112

To: <sip:Sub1@130.237.15.238>;tag=w8Wu

SIP to address: sip:Sub1@130.237.15.238

SIP tag: w8Wu

From: <sip:ccsleft@130.237.15.238>;tag=d6bf04da8d94fdfca282a7583d137144-

SIP from address: sip:ccsleft@130.237.15.238

SIP tag:
d6bf04da8d94fdfca282a7583d1371
44-edd5

CSeq: 2 NOTIFY

Sequence Number: 2

Method: NOTIFY

Event: location

Message body

Content-Length: 0

COS/CCS 2007-27

www.kth.se