Secure Context-Aware Mobile SIP User Agent

BEMNET TESFAYE MERHA

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2009

TRITA-ICT-EX-2009:63

Secure Context-Aware Mobile SIP User Agent

Bemnet Tesfaye Merha

July 5, 2009

Home University Supervisor & Examiner: Prof. G. Q. Maguire Jr.
Department of Communication Systems (CoS),
Royal Institute of Technology (KTH), Sweden

Host University Supervisor & Examiner: Prof. Rolv Bræk
Department of Telematics (ITEM),
Norwegian University of Science and Technology (NTNU), Norway

Preface

The work depicted in this report was carried out at the Wireless@KTH laboratory of Royal Institute of Technology (KTH), in Sweden. The major goal of the project is to examine how multimedia communication systems could adapt to the user's context. The focus of the practical part of this thesis project was on the design, implementation, and evaluation of a context aware Secure Session Initiation Protocol (SIP) user agent. This thesis is submitted to Department of Communication Systems (KTH), in Sweden, and to Department of Telematics (NTNU), in Norway in partial fulfillment of the requirements for a master's of Science degree in Mobile Computing and Information Security.

Acknowledgment

I would like to express my deepest gratitude to my supervisor professor Gerald Q. Maguire Jr. at Royal Institute of Technology (KTH) for his continuous support and guidance throughout the various stages of the project. Without his personal involvement and intervention at critical stages, it would have been very challenging to complete the project according to the initially set plan. His was of tremendous help in setting up the test beds. He provided practical and useful feedback throughout the project. He provided critical and useful suggestions on how to approach the research problem systematically and tactfully. Professor Maguire's continuous motivation and encouragement made my stay at the Wireless@KTH lab an enjoyable experience.

I would also like to thank my host university supervisor Professor Rolv Bræk in Norwegian University of Science and Technology (NTNU) for his helpful suggestions during the initial phase of the project. I would also like to thank Professor Mark Smith for providing me a Wasa board and the IR beacons used in this project.

Moreover, I would like to thank the NordSecMob consortium and the European Commission for funding my study and giving me the opportunity to participate in the NordSecMob program. Special thanks go to the program coordinators Eija Kujanpaa, May-Britt Eklund-Larsson, and Mona Nordaune for their helpful advice that made my stay a successful one.

Last, but not least, my family and friends in Ethiopia deserve special thanks for their unconditional support and encouragement throughout the past two years. I thank my parents, Tesfaye Merha and Elsabet Woldeselasse, for believing in me and opening their communication channels for words of wisdom while I have been away from home.

Abstract

Context awareness is an important aspect of pervasive and ubiquitous computing. By utilizing contextual information gathered from the environment, applications can adapt to the user's specific situation. In this thesis, user context is used to automatically discover multimedia devices and services that can be used by a mobile Session Initiation Protocol (SIP) user agent. The location of the user is captured using various sensing technologies to allow users of our SIP user agent to interact with network attached projectors, speakers, and cameras in a home or office environment.

In order to determine the location of the user, we have developed and evaluated a context aggregation framework that gathers and analyzes contextual information from various sources such as passive infrared sensors, infrared beacons, light intensity, and temperature sensors. Once the location of the user is determined, the Service Location Protocol (SLP) is used to search for services. For this purpose, we have implemented a mobile SLP user agent and integrated it with an existing SIP user agent. The resulting mobile SIP user agent is able to dynamically utilize multimedia devices around it ***without*** requiring the user to do any manual configuration.

This thesis also addressed the challenge of building trust relationship between the user agent and the multimedia services. We propose a mechanism which enables the user agent authenticate service advertisements *before* starting to redirect media streams.

The measurements we have performed indicate that the proposed context aggregation framework provides more accurate location determination when additional sensors are incorporated. Furthermore, the performance measurements indicate that the delay incurred by introducing context awareness to the SIP user agent is acceptable for a small deployment such as home and office environment. In order to realize large scale deployments, future investigations are recommended to further improve the performance of the framework.

Keywords: *SIP, context-awareness, service discovery, trust establishment*

Sammanfattning

Att vara medveten om kontexten är en viktig synpunkt av präglande och allestädes närvarande uppskattning av omgivningen. Genom att utnyttja den kontextuella informationen som samlats in från omgivningen, kan applikationen anpassas till användarens specifika situation. I denna avhandling använder man användarens sammanhang för att automatiskt upptäcka multimedia utrustning och tjänster som kan användas av en mobil Session Initiation Protokoll (SIP) användaragent. Placeringen av användaren mäter man med hjälp av olika sensorer för att låta användare av vår SIP användaragent att interagera med nätverk tillkopplat projektorer, högtalare och kameror i hem eller kontorsmiljöer.

För att avgöra var användaren befinner har vi utvecklat och utvärderat en sammanhangsstruktur som samlar in och analyserar innehållsbaserad information från olika källor; passiva infraröda sensorer, infraröd beacons, ljusstyrkan och temperaturgivare. Efter bestämmaning användarens placering den så kallade Service Location Protocol (SLP) användas för att söka efter tjänster. För detta ändamål har vi genomfört en mobil SLP användaragent och integrerat denna med ett befintligt SIP användaragent. Den resulterande i mobil SIP användaragent som dynamiskt kan utnyttja multimedia utrustning runt omkring utan att kräva att användaren skall kunna göra någon manuell konfiguration.

Avhandlingen tar även upp den utmaningen som krävs för att bygga förtroende mellan användaragenten och multimedia tjänster. Vi föreslår en mekanism som gör det möjligt för användaragenten att verifiera tjänstannonsering innan man börjar dirigera medieströmmar.

Dessutom så indikerar mätningarna av prestanda att fördröjningen som man utsätter den för genom att introducera ”medvetenhet om sammanhanget” till SIP användaragenten är acceptabel på hemma eller i en företagsmiljö. För att stora spridningar skall bli verklighet så rekommenderas det att göra mer forskning för att förbättra prestanda.

Nyckelord: SIP, kontext medvetenhet, tjänst upptäckt, förtroende etablering

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

3PCC	Third Party Call Control
ADC	Analog to Digital Convertor
AES	Advanced Encryption Standard
CA	Certificate Authority
CE	Compact Edition
CN	Correspondent Node
DES	Data Encryption Standard
IR	Infrared
IrDA	Infrared Data Association
MAC	Message Authentication Code
MIKEY	Multimedia Internet KEYing
PDA	Personal Digital Assistant
PIDF	Presence Information Data Format
PIR	Pyroelectric InfraRed
PKI	Public Key Infrastructure
PSTN	Public Switched Telephone Network
PUA	Presence User Agent
RFID	Radio Frequency Identification
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
S/MIME	Secure /Multipurpose Internet Mail Extensions
SA	Service Agent
SDP	Session Description Protocol
SER	SIP Express Router
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SLP	Service Location Protocol
SPI	Security Parameter Index
SRTCP	Secure Real-time Transport Control Protocol
SRTP	Secure Real-time Transport Protocol
TLS	Transport Level Security
UA	User Agent
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
VCP	Virtual COM Port
VOIP	Voice Over Internet Protocol
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

1. Introduction

This thesis examines how to enable multi-media communication systems to adapt to a user's context. The focus will be on the design, implementation, and evaluation of a context aware secure Session Initiation Protocol (SIP) user agent [1; 2]. Depending on the user's location and the availability of multimedia devices, our user agent will enable users to experience a better multimedia session ***without*** requiring the user to manually re-configure their session. For instance when a user in an ongoing video conferencing session via their Personal Digital Assistant (PDA) moves into a room having more powerful (or more capable) input/output multimedia device(s), the user agent should be able to send video streams to a network-attached projector while streaming the audio to network-attached speakers, and use a high definition camera installed in the room for video input. The problem here *is* ***not*** redirecting the media stream to different devices (as this is already demonstrated in previous thesis project), rather the problem is to facilitate the discovery of devices and services provided in a secure and transparent mechanism. Without knowledge of these multimedia devices there is no way to transfer the media streams to these devices and without trust there is no reason to believe that transferring the media streams to these devices is appropriate.

In order to understand the context of the user, we built upon a number of earlier thesis projects conducted in the Department of Communication Systems here at KTH [3; 4; 5; 6; 7]. For example, we will use of the sensor system developed by Daniel Hübinette to sense the occupancy of a room. His system, discussed in section 3.2, uses passive infrared detectors to sense directional movement of a heat source. A SIP presence user agent publishes occupancy information about this room to a context server. Interested applications can subscribe the status updates and will be notified when the occupancy of the room changes. By using this presence framework, our SIP user agent can learn that there is no one in the room, thus no one should object if the audio and video outputs are migrated to the room's audio system and video projectors (see Figure 1). The determination that the user was the only person in the room was possible because his or her SIP presence user agent differentiates between the presence of zero, one, or more than one person in a room. However, it does not address the problem of *identifying* the individuals in the room. In this case, it is important to recognize the identity of the user who just entered the room so that his or her presence information can be updated in the user's context server. Note that here we have referred to the user's context server as opposed to the room's context server. This distinction is important in terms of maintaining the personal integrity of the user.

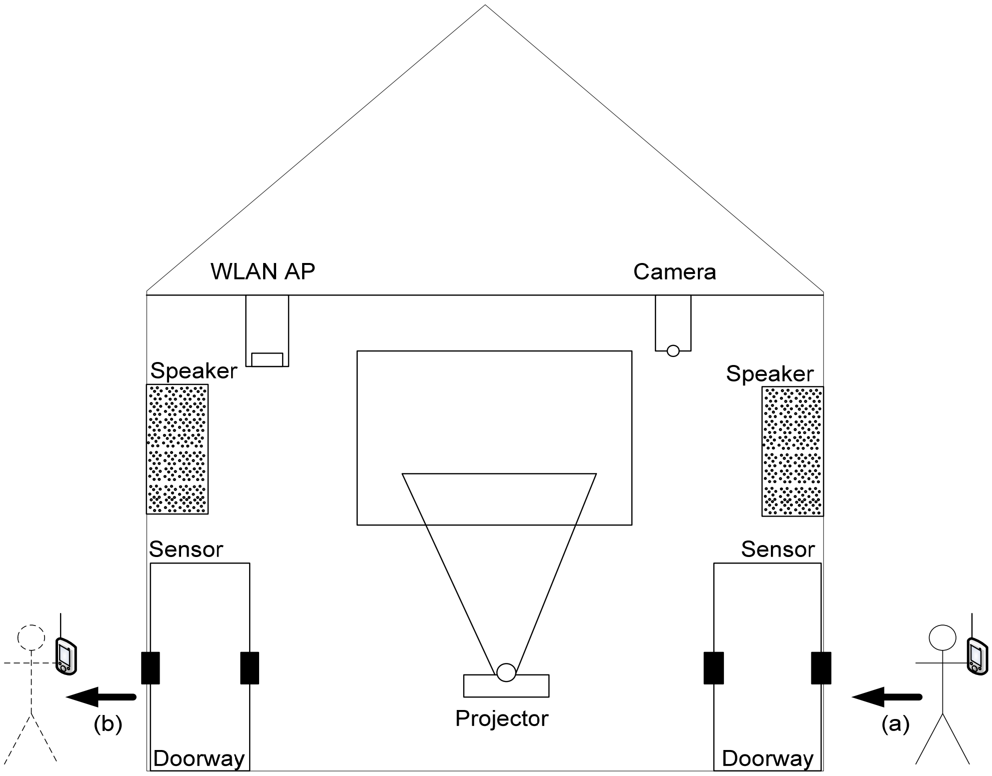


Figure 1: Overview of our system. As the user with a PDA moves into a room (a), the sensors installed on the doorway sense that the user has entered the room. The incoming video steam which was originally being displayed on the PDA will be transferred to a data projector and the sound will be steamed to a high quality set of speakers. As the user leaves the room (b) the streams will be redirected to the user's PDA.

An alternative solution would to utilizing RFID (for example, via an RFID tag). Today most individuals have RFID tags with them all the time. For example, the Stockholm Local (SL) transit system has started distributing RFID enabled smart cards as transit cards and in Trondheim city the bus operating company uses RFID enabled smart cards. Therefore an alternative solution could use a sensor system that can read such tags and update the status of the corresponding user using the unique serial number of the tag. Note that this requires a mapping to be established between a RFID tag's serial number and a SIP URI (this URI could identify the SIP proxy of a given user or another trusted third party who acts on behalf of this user or potentially directly identifying the user). However, directly identifying the user is fraught with many problems concerning the user's personal integrity. Alternatively, the room could have the RFID tag and the user's device could be equipped with a RFID reader - thus permitting the room to identify itself via a URI that can be used to get more information.

Once we are able to recognize the presence of the user in a specific room/location, the next question will be how our user agent learns about the devices and services around us. One solution could be to have all details related to the address (URL), type of service, and availability information preconfigured in the device. We envision our system enables a mobile user to have an enjoyable user experience, thus when the user walks into the room the list of available multimedia devices will be made available automatically. However, this requires either a mechanism to register devices and services in a registry which is available via the network or dynamic discovery of services via the user device's network interface[1]. The Service Location Protocol (SLP) is a service discovery protocol standardized by the Internet Engineering Task Force in their request for comments (RFC) 2608 [8]. SLP utilizes three types of entities: Directory Agent, Service Agent, and User Agent. Our main task regarding service discovery will be to understand how this protocol can be used to dynamically discover devices and services that are available to the user and to develop and evaluate a prototype to verify the design choices made.

The above scenario clearly raises a number of questions regarding trust. For example, how does the user know the devices and services being made available to them are devices and services that they can trust and not some rogue device that would like to eavesdrop their conversation or even worse a malicious service? We will examine various existing trust negotiation mechanisms [9; 10]. The trust model to be proposed should enable a user agent to dynamically build trust in devices that it has never used before without requiring the user to do significant manual configuration[2]. Obviously, it is important to support different levels of trust, ranging from not trusted, partially trusted, and trusted. Additionally devices and services can move up in the trust hierarchy as the user agent over time builds up confidence in the service. However, it is equally important to understand that the scope of trust we are trying to achieve has rather limited scope. This is practical since our system shall be used in a smart home or office environment, where the rate of exposure to new unknown devices or services is relatively low. Therefore instead of using complex trust negotiation and reasoning techniques based on distributed policies and ontology, we will be interested in a simple but sufficiently secure trust negotiation model.

The rest of the thesis is organized as follows. Chapter 2 will summarize the basic concepts and technologies which serve as a foundation for our work. Chapter 3 will present a summary of related work previously conducted in the area of context-aware computing and multimedia communication. Chapter 4 starts by presenting the requirements of our context-aware framework and continues by discussing various design and implementation issues related to the framework. Chapter 5 will present the proposed device discovery, trust establishment, and media redirection mechanism. Chapter 6 will present the results obtained when evaluating the context aggregation framework and the media redirection technique followed by detailed discussion of the findings. Finally chapter 7 contains some concluding remarks and recommendations for the future.

[1] Note that here we are assuming that the device is not doing the discovery via some other means such as reading RFID tags, scanning for bar code or other visible markers, etc.

[2] Ideally the user should only need to specify a general trust profile once - with the option in specific cases to say: Do not use that device or service again.

2. Background

In this section we summarize basic concepts and technologies which we believe are valuable to our research. A critical analysis of the related work follows this.

2.1 Context Aware Computing

Context aware computing refers to systems that sense and react based upon their environment. Such systems may provide customized service with minimal user interaction based upon context information sensed from the environment. It is important to understand that the kind of information that can be considered as part of this context depends on the application and can take may different forms. As discussed by Dey [11], initially context awareness was perceived to be the user's location. However, systems that can take various aspects of the user's context including voices, light levels, weather conditions, presence of people around us, availability of computing resources and network connectivity, communication cost, and bandwidth have been developed. Thus it is clear that a general model to support context aware systems is both useful and necessary.

The survey by Bolchini, *et al.* [12] provides a data driven comparison of several general purpose models. In this survey sixteen different frameworks are compared based on aspects such as location, time, previous context history, user profile, and so on. The survey also assesses the way each of these models build, manage, and exploit context. Although it is necessary to have a general framework with which application developers can build context aware systems, it has been impractical to aim for a model which will fit every application domain. Instead the most practical approach is to consider using a model that fulfills the primary design goals with appropriate adjustments. In the design phase of our thesis we will identify an appropriate model that meets the requirements of our context aware SIP user agent.

2.2 Dynamic Service Discovery

Dynamic service discovery enables potential consumers of a service to dynamically discover a service that matches a required service description exists and to learn how to communicate with this service. In mobile computing an intelligent way of discovering services has a special value as the available services change with location and the lifecycle of services (i.e., new services are created, fielded, operate, and are terminated). For instance someone who just moved into a new office should be able to issue a print request that requires the use of a color printer, which can print 14 pages per minute and is within a reasonable distance of the user. Additionally, it should be possible to utilize such a printing service without being required to configure and install every printer in the whole building into the user's device(s).[3] Consider a mobile SIP user agent which is able to utilize available multimedia devices such as projectors, speakers, and cameras in a room without manual configuration. Our major task related to service discovery will be to analyze some of the

[3] For an example of such a service, see the work of Athanasios Karapantelakis [56].

existing protocols and technologies and implement and evaluate our choice of a service discovery mechanism in our user agent. In the following subsections we summarize some of the candidate technologies considered in our project.

2.2.1 Universal Plug and Play (UPnP)

UPnP is a standard based upon existing networking protocols designed to provide seamless installation and configuration of devices in home and office environments. The UPnP Forum is responsible for developing and maintaining the standard. This forum has more than 875 members (including major hardware, operating system, and application vendors). UPnP uses established networking protocols and technologies (specifically TCP/IP, UDP, HTTP, and XML) to connect computers, home appliances, and wireless devices together [13].

Every device in UPnP is identified using an IP address and should implement a DHCP client to obtain an IP address dynamically. When a new node joins the network it starts by advertising its services to a special node called a control point. Similarly when a new control point joins the network it starts by looking for devices around it. All UPnP messages are formatted in an XML based device template. This template specifies the capabilities of the device. Once a control point learns about the capability of the device, it can send queries to obtain more details about the device or can subscribe for status updates of the device or its services. Often the control point can present all the details and status updates of the devices using a graphical user interface viewed in a browser.

It is important to understand that such a graphical user interface is not suitable for our requirement, as we do not want the user to have to manually configure their device in order to make use of at least specific subclasses of multimedia devices (projectors, speakers, cameras, etc.) via their SIP user agent.

2.2.2 Jini

Jini is another technology for discovering services in a distributed computing environment. This technology was originally developed by Sun Microsystems and now it is in an incubation period to be passed to the Apache Software Foundation under the name River [14]. Jini is based on Java and provides a device discovery service based upon Java Remote Method Invocation (RMI). Jini not only provides service discovery, but by providing a set of Java methods to utilize the service it provides increased functionality by allowing applications to utilize resources distributed over a network as if they were installed locally. However, to utilize these methods requires that the device that wishes to invoke these methods have a Java virtual machine in order to execute the method(s). It should be noted that many people believe that this also means that the device offering services also needs to implement a JVM; however, this is false - as it simply needs to know how to implement the Java RMI and the service could be implemented in any language that the service creator wishes to use.

A Jini network includes three type of entities: clients, (one or more) servers, and a lookup service. A node that wishes to offer a service starts by discovering the lookup service, which is used to register the service to be provided in the logical Jini network.[4] If a lookup service is available, then the node wishing to register the service will obtain a Service Registrar object by which it can register its services. The clients follow a similar discovery procedure to obtain a Service Registrar object. Subsequently the client can request the lookup service to search its list of registered services based on name, type, or description of a service. Upon finding a match the lookup service returns a Java proxy enabling the client to directly connect to the server. The communication between the three entities is enabled by using Java serialization to transport Java objects between Java virtual machines.

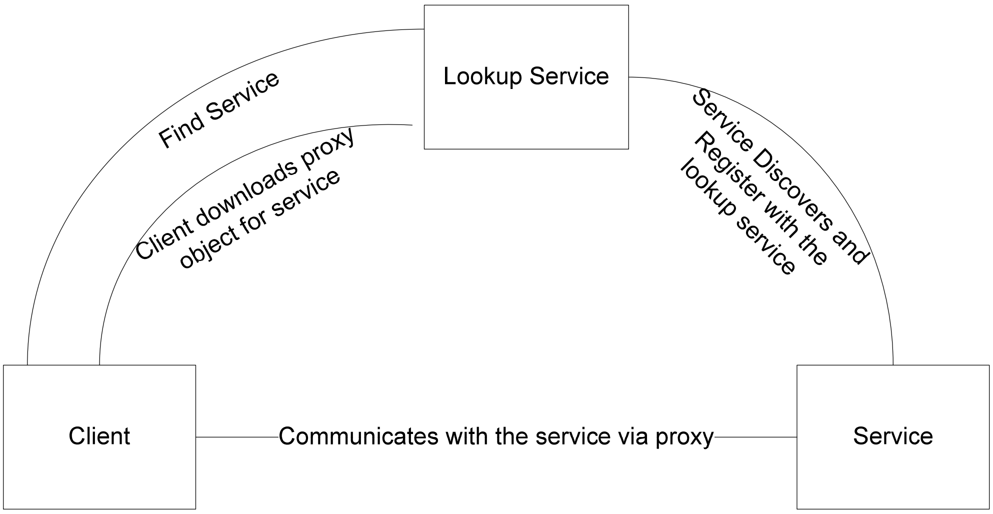

Lookup Service
Find Service
Client downloads proxy object for service
Service Discovers and Register with the lookup service
Client
Communicates with the service via proxy
Service

Figure 2: Service discovery using Jini

Chen *et al.* [15] give a critical evaluation of Jini in the area of mobile computing environment. The major weakness identified is using Java class interfaces for matching service requests. This approach is also poor for representing complex service descriptions. It turned out to be difficult to represent service requests using the interface based syntax and a better solution would utilize a more flexible way of representing services using XML making the look up process more adaptable. Because our system requires more flexible querying mechanism, Jini is not an ideal solution for our service discovery problem.

2.2.3 Service Location Protocol (SLP)

SLP [8] introduces a dynamic service discovery protocol that can be used by devices in an IP network. It allows hosts to discover devices and services without prior configuration. The protocol specifies three entities: Directory Agent (DA), Service Agent (SA), and User Agent (UA). The Service Agents advertise details of available services to interested user agents. Directory agents can be used to increase scalability by allowing service advertisements from service agents to be stored so that interested user agents can perform a

[4] Note that we said the "logical Jini network" as it is the ability to access the lookup service that defines this network - not the physical network.

lookup using a DA rather than having to learn about all the SAs by itself. A SLP UA issues Service Requests on behalf of a client application. These service requests can either be multicast to SAs or unicast (if the UA knows a DA in the network). In both cases, the UA receives a service reply which it can use to contact the service. In order to lookup a service or register new services both the UA and SA need to know of a DA. This is done by broadcasting or multicasting[5] a service request, in response an active DA (or an SA) will reply with service reply. It is important to note that the presence of a DA is optional - as it is possible to use SLP simply with SAs and UAs.

Every service on the network is identified using a URL for example, the URL *service:printer:lpr://myprinter/myqueue* describes a printing service provided by a printer named *myprinter* that uses the Line Printer Remote (LPR) protocol. Optionally URL services can have any number of attributes specified using a name-value pair. These attributes will be used to describe the details of the service and can be used by the user agents when issuing a service request query. SLP allows grouping of services based on location, administrative structure, or proximity in network topology. If required, user agents can be assigned to a given scope; in which case they will only be able to discover services in their scope [8]. It is also important to note that SLP URLs can include IP addresses rather than names - so the network infrastructure does not need to have an available DNS server.

In this project we demonstrate, how one can easily implement a simple SLP UA. The SLP UA is incorporated inside the SIP UA to allow us dynamically discover services and devices in the room (see section 5.4.3). The choice of SLP as a dynamic service discovery protocols is based on the fact that SLP is intended to function within a network under cooperative or personal administrative control. SLP relies on networking features such as multicast routing, organization of clients and services in to a group, and implementation of security policies, all which are not suitable for a global scale deployment. However, our system is intended to be used in a home or office environment where such administrative control is readily available, making it easy to realize SLP functionality. Furthermore, compared to other dynamic service discovery protocols SLP provides more flexible service queries - using LDAPv3 predicate logic [8].

2.3 Session Initiation Protocol (SIP)

SIP is a signaling protocol used to establish, modify, and teardown a multimedia communication session over the internet. SIP was initially developed by IETF’s Multiparty Multimedia Session Control (MMUSIC) working group, and then taken over by IETF’s SIP working group [16]. Currently RFC 3261 describes the core functionalities of the protocol [1]. The protocol was designed with the intention to add signaling and call setup functionalities to an IP based network. Although these functions have long been present in the traditional Public Switched Telephone Network (PSTN) systems, a significant difference between a SIP based multimedia system and the PSTN is due to their extreme difference in design principles. PSTN networks the use Signaling System 7 (SS7) protocol for call setup and call

[5] For IPv6 a set of multicast group IDs are defined and broadcast only SLP configurations are not supported under IPv6.

processing. SS7 is a centralized approach and requires complex and intelligent equipment in the core network, and allows dumb terminals at the end points. Whereas SIP builds upon the basic internet principles, where the network is dumb and the end-points have significant computing capabilities

The fact that a SIP network places all of the computationally intensive operations at the end-points enables us to achieve high scalability; while at the same time delivering a wide variety of end-to-end services. The success of SIP can be seen in its usage in various multimedia communication systems - including voice and video conferencing, presence applications, instant messaging, collaboration applications, and file sharing systems.

In order to realize all these services, SIP utilizes various other protocols. It is important to note that SIP is only meant for the signaling portion of a multimedia communication. SIP uses a separate protocol called the Session Description Protocol (SDP) for describing the media content of the session to be established. Information including the IP address, port number, and the type of CODEC to be used for each media stream is included in the SDP embedded in the body of the SIP messages. SDP introduces a negotiation scheme between endpoints in order to agree upon a common media type and format - with available CODECs. In the traditional PSTN network such flexibility is not available as user terminals do not have the ability to negotiate what media types, formats, and CODECs to use. As a result the PSTN generally offers only a very limited set of services to the end-points and these services generally have a fixed quality (in fact, the emphasis of the PSTN has been on guaranteeing a fixed QoS).

In this thesis SDP's ability to efficiently redirect an ongoing SIP session to a different terminal will be exploited. More specifically, we will investigate the approach suggested by Oscar Santillana in his master's thesis (see section 3.3 on page 24). A careful investigation of this approach along with other alternatives will be conducted in order to find an efficient media redirection scheme. Note that media redirection is a central element of this thesis project as we wish the user to easily be able to exploit local input/output devices without requiring extensive manual configuration - hence requiring the user to initiate or receive a new session is not acceptable.

Based on the agreed media type(s), format(s), and CODEC(s) the Real-time Transport Protocol (RTP) [17] will be used to carry the actual media content. RTP encapsulates audio and video samples along with a sequence number, timestamp, and an information about the media sources (this additional information is included in the RTP header [2]). RTP can use statistical feedback provided by the RTP Control Protocol (RTCP) [17] to adapt the quality of the media stream to network conditions. RTCP periodically transmits control packets containing information about the stream (including bytes sent, lost packets, jitter, and roundtrip delay) which can be used by the receiver to enhance the quality of the multimedia stream or used by the sender to adapt its transmission of a multimedia stream. It is important to note that multiple media types, formats, and CODECs can be negotiated, thus the application can use any of these during the session [18].

2.3.1 Components of a SIP network

A SIP network consists of SIP user agents and a variety of SIP servers. Each of these will be described below.

SIP users are addressable entities that participate in SIP sessions. Users are identified with a Universal Resource Identifier (URI), similar in format to an e-mail address. A SIP URI has the general form *sip:name@domain:port* where *name* is the name of the user; *domain* is the fully qualified domain name of the user's proxy server, and *port* is the port number where the proxy server is listening for a connection (the default is 5060). A SIP URI can also be used to address users with an E.164 phone number. For example, the URI of the form *sip:+46-700-680-137@gateway.com* may refer to a voice mailbox of a user. However, in order to use E.164 phone numbers a simple DNS lookup is done to find the address of the gateway between the SIP network and the PSTN that is associated with this E.164 number. The DNS query can return a variety of answers ranging from the IP address and port number of a media gateway to a new URI that is to be used.

SIP user agents (UAs) are end-points used for sending and receiving of SIP messages. User agents can be either implemented in hardware (for example, a dedicated analog telephony adapter, a SIP phone, or similar device) or as a soft-phone running on a general purpose computer or handheld device. Alternatively SIP user agents can also be implemented as a gateway to another network; for instance as a gateway to a PSTN network. SIP user agents have two basic functions: initiating SIP requests and receiving and responding to requests. The part of the user agent that generates requests is called a user agent Client (UAC) and the component that responds to requests is called a user agent Server (UAS).

Each SIP user agent requires at least one valid IP address (usually obtained from a DHCP server).The UA should be able to resolve domain names using a DNS server, and so on. Users of SIP systems, with a fully qualified URI, are associated with a user agent upon registering with their registrar server (see below). The user agent will be able to receive an incoming invitation to a SIP session once its current location is known to the registrar server. It is important to note that the called user's SIP proxy can be used to implement the callee's call preferences; thus these preferences can be processed before the user's UAS is actually contacted.

In this thesis we will use the Minisip [19] user agent both for handheld devices and for stationary systems. Minisip is an open source SIP user agent being developed at KTH (together with others). It is written in C++. Minisip is an ideal choice for our project for number of reasons. It has been developed in a research environment where the main focus has been providing end-to-end security; hence implementations for TLS, SRTP, MIKEY, and other security protocols are provided. Ports of Minisip to different platforms include the HP iPAQ PDAs, Microsoft's Windows XP, and a variety of Linux, and UNIX systems. Appropriate extensions of functionality will be investigated to incorporate dynamic service discovery and media redirection techniques along with security solutions to be able to build trust relationships with devices near the Minisip user agent.

Although SIP user agents can communicate in a peer-to-peer manner, it is convent to use a central network element to help user agents to easily setup SIP sessions. A **SIP proxy** is such an entity, as it helps route a SIP user agent's requests to the destination user agent. Incoming invitations to a user agent to join a session are forwarded to the destination user agent according to the preferences set earlier by the user. Besides introducing a great deal of flexibility in the overall system, SIP proxies also provide a mechanism to perform a number of security functions, such as authenticating user agents and authorizing services to the users.

A redirect server is a user agent server that responds with 3xx messages for each request it receives. Upon receiving such a response the originator of the request will make a new request using the SIP URI received in the 3xx message. The main reason for utilizing redirect servers is to reduce the load on proxy servers, which otherwise are responsible for routing SIP requests.

In order to receive incoming session invitations SIP user agents must register their contact information with a **registrar server**. This is done by sending a REGISTER request to a registrar server. The registrar server uses a location server to store the contact information associated with a SIP user. Note that what it is actually storing is a Fully Qualified Domain Name (FQDN) or IP address that can be used to contact one or more SIP user agent servers. Thus one has to be careful about the use of the term *location* as the registrar need not know the physical coordinates of a user agent.

In a SIP network a **location server** provides a database that can be used to store information related to users' contact information, IP addresses and port numbers. SIP user agents do not directly access this information; rather it is updated and retrieved though their respective proxy and registrar servers. The interaction with the location server is not defined in the SIP RFC and is done using a non-SIP protocol. In the case of this thesis project, the SIP Express Router (SER) will be configured to use a MySQL server for storing all the information related to user preferences and their registered locations [20].

2.3.2 SIP Dialogs and Transactions

It is sometimes confusing to clearly differentiate between SIP dialogs and transactions. It is important to understand the difference in order to correctly implement the session transfer mechanism presented in section 5.3.

- **Dialog:** A dialog (previously called a *call leg*) represents a peer-to-peer relationship between user agents that persist for some time. It is used to properly sequence message proper routing of requests between these peers[6]. A dialog is identified using a dialog identifier consisting of a Call-ID, a local tag, and a remote tag. A dialog is created when a request gets a non-failure, final response (2xx and 101-199 responses with a "To" tag). If a request gets a non-final response, it is considered as an early dialog.
- **Transaction:** represents a set of messages between peers, starting from a request from a client to a final (i.e., non 1xx) response from a server. As shown

[6] Note that SIP requests can also be processed *outside* a dialog, in which case the individual requests will establish a dialog.

in Figure 3, an INVITE transaction includes the INVITE, 180 Ringing and 200 OK messages. Note that if an INVITE request gets a final response, then the ACK is considered as separate transaction. We can also observe that a set of transactions can be part of a dialog. The main purpose of maintaining state about transaction is to properly deliver requests to the Transaction User (TU). For instance a client transaction is responsible for receiving responses, filtering out retransmissions and delivering it to TU.

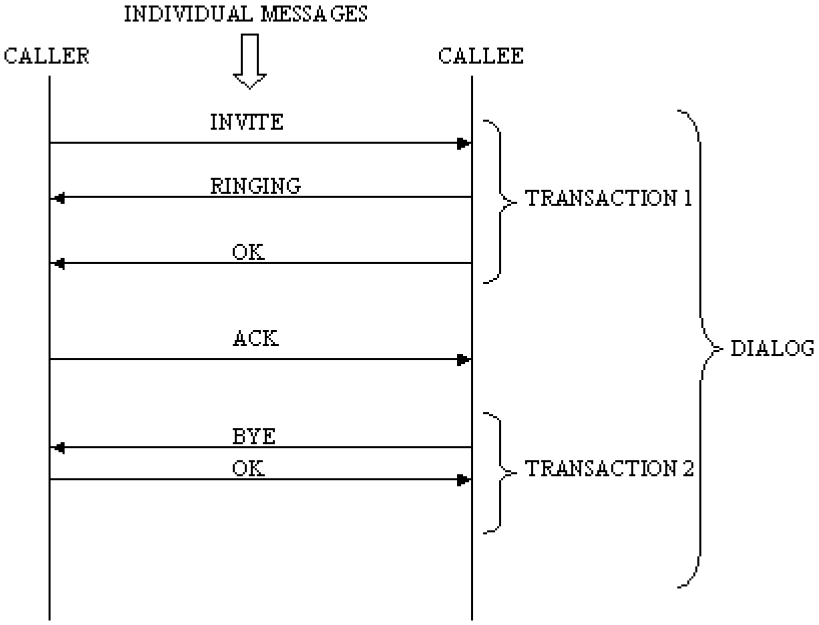


Figure 3: Relationship between SIP dialog and transaction

2.3.3 Real time Transport Protocol (RTP)

RTP is an application layer protocol that is designed to provide real-time transport of audio and video data over an IP network [17]. Majority of RTP implementations are based on UDP instead of TCP. This is because multimedia applications require timely delivery over reliable delivery. The latency involved in establishing connection and retransmitting missing packets makes TCP unsuitable for real-time transport. Instead, RTP uses UDP and adds various functionalities such as sequencing of packets, jitter control and error concealment for lost packets.

RTP is used with Real-time Transport Control Protocol (RTCP) [17]. RTCP provides out of band control information for RTP stream. The primary function of RTCP is to provide a feedback on the quality of service of the media stream by periodically sending statistical information to the session participants. RTCP report includes information such as transmitted packet counts, lost packet counts, jitters, and round trip delays. Applications use this information to control transmission behavior by adjusting flow rates or changing CODEC used.

2.4 Secure Multimedia Communication

When using SIP based IP telephony, very few users pay attention to security. Compared to Public Switched Telephone Network (PSTN) based telephony systems, SIP based systems suffer from numerous security concerns. These concerns are not a result of flaws in SIP or other supplementary protocols; but because of the availably of wide selection of tools to perform serious attacks on an IP network. The following are a list of some of the security concerns that are straight forward to apply to a SIP based system; therefore one must consider the appropriate counter measures.

Table 1: Security threats related to IP telephony and corresponding protection mechanism (A subset of those security threats presented in table 9.1 on page 160 of [2]).

Threat	Description	Protection Mechanism
Session Hijacking	User dials a SIP URI, but actually establishes a session with another user.	Authentication of signaling.
Registration Hijacking	Incoming calls to a user are diverted to a third party.	Integrity protection of registration.
Impersonation	A third party impersonates another user in a session.	Using enhanced SIP Identity
Eavesdropping on signaling	A third party tracks and records with whom a user is communicating with by monitoring SIP messages.	Using TLS
Eavesdropping on media	A third party tracks and records media streams	Using SRTP
Session disruption	Calls to or from a user are disrupted after they are established.	Integrity protection of signaling.
Denial of service	Calls to or from a user are prevented.	IP, SIP, and RTP layer traffic management using various techniques.

2.4.1 Secure Signaling

In order to prevent some of the threats described above, SIP utilized various techniques of providing confidentiality and integrity protection. Instead of defining a new a security mechanism, SIP utilizes existing security protocols operating in different layers. Below we describe some of the widely used techniques.

2.4.1.1 Using network and transport layer security

In order to provide network layer security we can use IPSec. IPSec is commonly used in architectures consisting of hosts that are in an administrative domain where there is an existing trust relationship with one another. IPSec is usually implemented in the host operating system or on a security gateway to provide confidentiality and integrity protection of all network traffic on a particular interface [1]. This means IPSec security has not direct interaction with SIP network elements like user agents, proxy, and registrar servers. This makes IPSec ideal for network architectures where introducing a security mechanism to these SIP entities is not desirable.

Transport Level Security (TLS) on the other hand provides a transport layer security for SIP messages. In comparison to IPSec, TLS is suitable when there is no preexisting trust relationship between two hosts. In a situation where two user agents do not have an existing trust relationship, TLS can be used to establish a hop-by-hop security using digital certificate chain. Once TLS connection is established, all the SIP messages will be confidential and integrity protected. However, it is important to understand that TLS can only prove hop-by-hop security, thus a user agent that sends request using TLS cannot be assured that TLS will be used end-to-end. For this purpose the secure SIP URI (*sips*) is defined. By using *sips,* all the requests made by the user agent are granted that all intermediate hops will use TLS. One exception is the last hop of the request, which could be protected using some other means (for instance IPSec or some lower layer security).

2.4.1.2 Using S/MIME

S/MIME is an alternative solution used to provide end-to-end security for SIP messages. S/MIME uses public key infrastructure to provide confidentiality and integrity protection of the SIP body [1]. A user agent that uses S/MIME encrypts the body of its SDP using the public key of the end user. In order to provide integrity protection the digital signature of the SDP is attached to the SIP message.

2.4.2 Media Security

The security measures discussed in the previous section provide protection for the signaling portion of a multimedia session. The RTP media (carrying the content in the session) can be protected using a separate protocol called *Secure RTP (SRTP)* [21]. SRTP provides privacy, authentication, and replay protection for the media stream. The detail of each of these is described below.

2.4.2.1 Encryption of RTP stream

SRTP uses AES (Advanced Encryption Standard) to encrypt/decrypt RTP packets. AES can be used with various key and block sizes. In SRTP a 128 bit block is encrypted with a 128 bit key. In order to encrypted larger block size, two[7] modes of operation - *Segmented Integer Counter Mode* and *f8-mode* are used. Table 2 presents the predefined security suites for both modes of operation and the corresponding key length. When used in counter mode, the key stream is generated by encrypting successive integers as follows.

$$KS = E(K_e, IV) \,||\, E(K_e, IV + 1 \bmod 2^{128}) \,||\, E(K_e, IV + 2 \bmod 2^{128})$$

[7] SRTP has a third cipher mode called NULL Cipher, which provides no encryption (i.e its output is identical to the input payload)

Where Ke is the encryption key, E() is the AES encryption function, and (IV) is calculated as follows

$$IV = (k_s * 2^{16})\ (SSRC * 2^{64}) \oplus (i * 2^{16})$$

Where k_s is the session salting key, i is the SRTP packet index, and SSRC is the synchronization source. In this mode encryption is based upon XORing the RTP payload with the generated key stream.

An important point to note here is that reuse of the keystream must be avoided. If a keystream is used more than once a trivial attack as shown below can be realized easily.

$$C_1 = KS \oplus P_1$$

$$C_2 = KS \oplus P_2$$

Where C is cipher text, KS is the key stream, and P is the payload. The attacker can compute:

$$C_1 \oplus C_2 = (KS \oplus P_1)\ XOR\ (KS \oplus P_2) = P_1 \oplus P_2$$

Now if the attacker can decrypt C_1, P_2 can be obtained as follows,

$$P_1 \oplus (P_1 \oplus P_2) = P_2$$

It is due to this problem that the SRTP RFC mandates that, a key stream generated from the same index and key must ***never*** be used more than once. By including the packet index when computing the IV, SRTP generates a unique keystream per packet. Furthermore, SRTP allows sharing of the master key across different streams belonging to the same RTP session by including the SSRC in IV calculation.

The *f-8 mode* of operation uses the f-8 mode originally defined for data encryption in Universal Mobile Telecommunications System (UMTS) systems. This mode of operation is based on Output FeedBack (OFB) mode, where the output of each encryption block is feed as an input into encryption of the next block. It uses AES as a block cipher with the same block and key size as in the counter mode described above. More details about this mode of operation can be found section 4.1.2 of the SRTP RFC [21].

Table 2: Predefined security suites.

Suite Name	Encryption Key length(bits)	Authentication Key Length(bits)
AES_CM_128_HMAC_SHA1_80	128	80
AES_CM_128_HMAC_SHA1_32	128	32
F8_128_HMAC_SHA1_80	128	80

2.4.2.2 Authentication and Integrity Protection of RTP packets

The above method provides confidentiality for the media stream, but it does not prevent the attacker from forging RTP packets. SRTP provides a mechanism to authenticate individual packets thereby maintaining the integrity of the media stream. This is done by using a keyed hash function called HMAC-SHA1. The hash value is computed with the authentication session key (k_a) and a portion of the RTP header and the payload as shown in Figure 4. The HMAC-SHA1 produces a 160 bit output which is truncated to become 80 or

32 bit authentication tag appended to the SRTP packet by the sender. The receiver will compute the hash value similarly and verify if the authentication tag matches value computed locally. If it does, then the packet will be sent out for play out, otherwise it will be dropped.

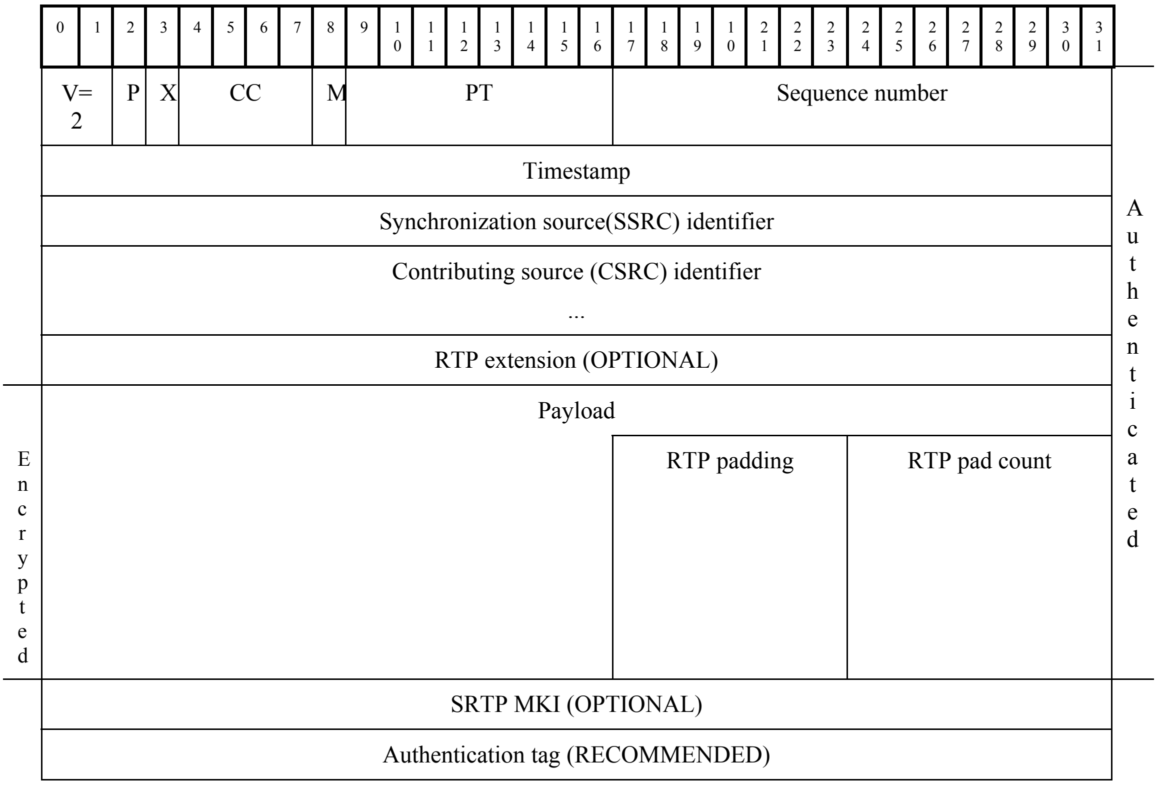


Figure 4: Format of SRTP packet([21])

2.4.2.3 Replay protection

With message authentication in place, the attacker cannot spoof the media stream. However, an adversary can still capture SRTP packets and re-inject these packets into the network later. For the victim to successfully play out the replayed packet the replayed packet should be re-injected before the authentication session key is renewed, otherwise the authentication would fail. SRTP provides a solution to avoid such a replay attack. The receiver keeps track of the last few sequence numbers that have been played out. Typically this will be done by using a sliding window of an acceptable range of sequence numbers. Any value less than this range will be assumed to be replay attempt and will be dropped. An important point to note here is that this replay protection works only if authentication is enabled. Otherwise the attacker will be able to spoof the sequence number without this being noticed by the replay protection mechanism.

2.4.3 Key Exchange

In order to provide end-to-end security, the communicating entities must agree upon a cryptographic keys and parameters. This is done by using key exchange protocols. In this section we will present two key exchange mechanisms used in this thesis project.

2.4.3.1 Multimedia Internet KEYing (MIKEY)

MIKEY is a key management solution that is used to exchange keys and related security parameters used to secure a real-time multimedia session. A single (exchanged) master key will be used to derive session keys (collectively known as TEK - Traffic Encrypting Keys) used for encrypting and integrity protecting the media stream. Although MIKEY is a general key exchange protocol, it has some features which make it an ideal choice for real-time communication systems (both for unicast and multicast scenarios). Compared to other similar key management protocols (for instance Internet Key Exchange - IKE), MIKEY has lower latency making it suitable for real-time communication systems [22].

The central goal of MIKEY is securely exchanging the master key, also called the TGK (TEK Generating Key). Therefore in order to exchange the TGK between the participants, the MIKEY messages[8] need to be encrypted and integrated protected. RFC 3830 [22]defines three different methods of securely transporting a TGK using a pre-shared key, public-key encryption, and Diffie-Hellman key exchange. In this thesis project MIKEY using Diffe-Hellman key exchange is used to establish a secure session between the correspondent node and the mobile node.

2.4.3.2 SDP Security Descriptions for Media Streams

A new media level SDP attribute called *crypto* attribute provides a mechanism to exchange key material and other security parameters for SRTP. Including this in the SDP enables the parties to agree upon a cryptographic suite, key parameters, and session parameters using either a single message or a round trip exchange [23]. In contrast to MIKEY, this approach is designed specifically for SRTP. Furthermore, the *crypto* attribute is only meant to establish a security context in a unicast scenario, while MIKEY could be used for both unicast and multicast cases. The syntax of the *crypto* attribute is given below.

```
a=crypto:<tag> <crypto-suite> <key-params> [<session-params>]
```

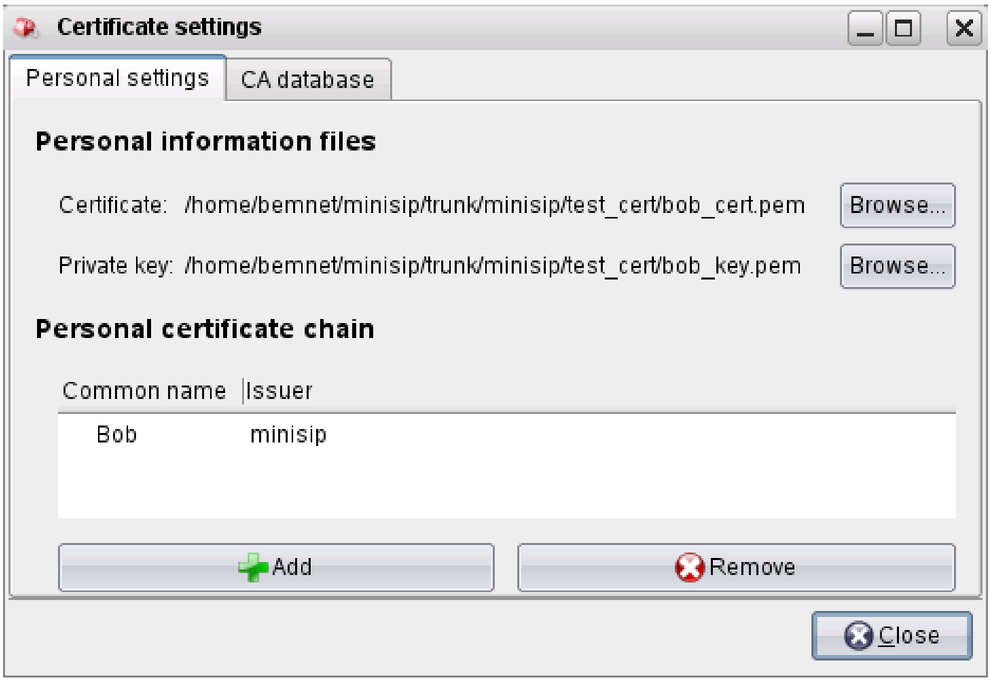
The *tag* is an identifier of specific *crypto* attribute. The *crypto-suite* is used to identify the encryption and authentication algorithm to be used for SRTP. An example *crypto-suite* is AES_CM_128_HMAC_SHA1_80, which uses AES in counter mode for encryption and HMAC with SHA1 for authentication. The *key-params* field provides the keying material to be used for the *crypto-suite*. It is base64 encoded octet string concatenation of the master key and the master salt used for SRTP. The session-params field is used to specify parameters, such as lifetime of keys and Master Key Index (MKI) number of SRTP packets.

Because the keying materials are carried inside the SDP message, this approach can only be used if the SIP signaling is protected. Thus the *crypto* attribute will only be used when the SIP messages are confidential and integrity protection is provided using either S/MIME or TLS as described in section 2.4.1. In our project the *crypto* attribute is used to transfer the master key from the mobile node to the local node as presented in section 5.4.5.

[8] MIKEY is a general protocol and it does not define which SIP messages will embed MIKEY messages.

2.4.4 Minisip support for security

Minisip [19] is an open source SIP user agent developed at KTH. The most attractive feature of Minisip is its support for security. It implements TLS to secure the signaling and SRTP for protecting the media stream. For key management Minisip implements MKIEY using pre-shared key, Diffie-Hellman key exchange, and certificate based encryption (see Figure 6). In order to use the last two options of MIKEY and TLS, the user agent must be configured with security certificates. Minisip supports X.509 certificates from both trusted Certificate authorities (CAs) as well as self-signed certificates. The screenshot in Figure 5 shows the configuration of a personal certificate chain for Minisip running on a Linux machine. For the iPAQ PDA there is no graphical user interface, thus this configuration is done using a configuration file (see Appendix L)



Certificate settings

Personal information files

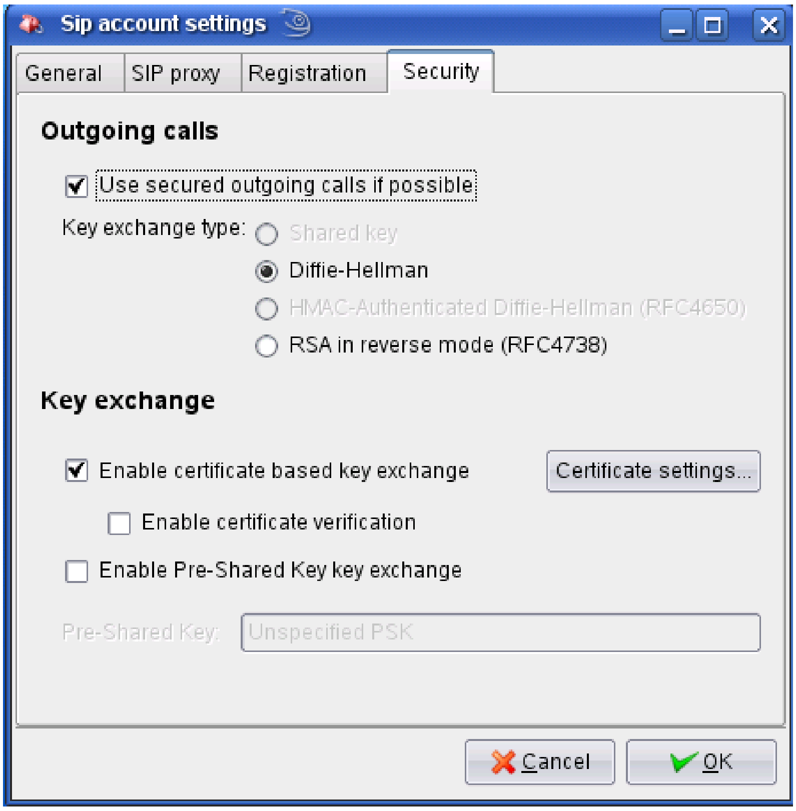
Certificate: /home/bemnet/minisip/trunk/minisip/test_cert/bob_cert.pem

Private key: /home/bemnet/minisip/trunk/minisip/test_cert/bob_key.pem

Personal certificate chain

Common name	Issuer
Bob	minisip

Figure 5: Certificate configuration in Minisip



Outgoing calls

☑ Use secured outgoing calls if possible

Key exchange type:
○ Shared key
● Diffie-Hellman
○ HMAC-Authenticated Diffie-Hellman (RFC4650)
○ RSA in reverse mode (RFC4738)

Key exchange

☑ Enable certificate based key exchange

□ Enable certificate verification

□ Enable Pre-Shared Key key exchange

Pre-Shared Key: Unspecified PSK

Figure 6: MIKEY configuration of Minisip

2.5 Trust Relationships

Due to various security concerns, it is important to establish an appropriate degree of trust between two entities before they attempt to perform an online transaction. On the internet today various services require two strangers to meet for the first time and conduct a business transaction. Such transactions could involve online purchases, access to confidential information, and, participating in a multimedia session. For these systems to function properly the participants must perform mutual authentication in order to make sure that they trust each other.

When properly implemented digital signatures gives the receiver sufficient reason to believe that the message was sent by the claimed sender. Compared to a handwritten signature, a digital signature is more difficult to forge. Digital signatures use a public key cryptography algorithm (sometimes called an asymmetric key algorithm). The distinguishing technique used in public key cryptography is that the key used for encryption is not the same as the key used for the decryption. Each user will have a cryptographic key pair – a so called public and private key. The private key is kept secret and is only known by the owner. In contrast, the public key is not a secret at all and can safely be widespread to allow easy public access. Messages are encrypted using the recipient’s public key and can only be decrypted using the corresponding private key. The security of public key cryptography relies on the fact that the two keys are mathematically related in such a way that the private key cannot be feasibly be derived from the public key.

In order to digitally sign a document, a one way hash functions are used to compute a fixed size representation of the entire document. Examples of hash functions used for digital signature include MD5, SHA-1, and HAVA. The signer uses his/her private key to encrypt the hash value. To encrypt the message asymmetric cryptographic algorithms such as RSA and DSA can be used. This encrypted hash value is what we call the digital signature. Before sending the signature to the verifier, the sender can attach a digital certificate signed by a trusted third part (usually a trusted Certificate Authority - CA) proving that the included public key belongs to the claimed identity (see Figure 7).

Upon receiving the digitally signed document, the verifier performs the exact reverse operation performed during signature generation. It uses the public key to decrypt the signature, which reveals the hash value encrypted by the sender. The receiver also computes the hash value of the message to be verified and compares it with the hash value obtained after decryption. If the two values match, then the verifier can be sure that the message was sent by a trusted individual (more specifically by someone who is in the position of the private key)

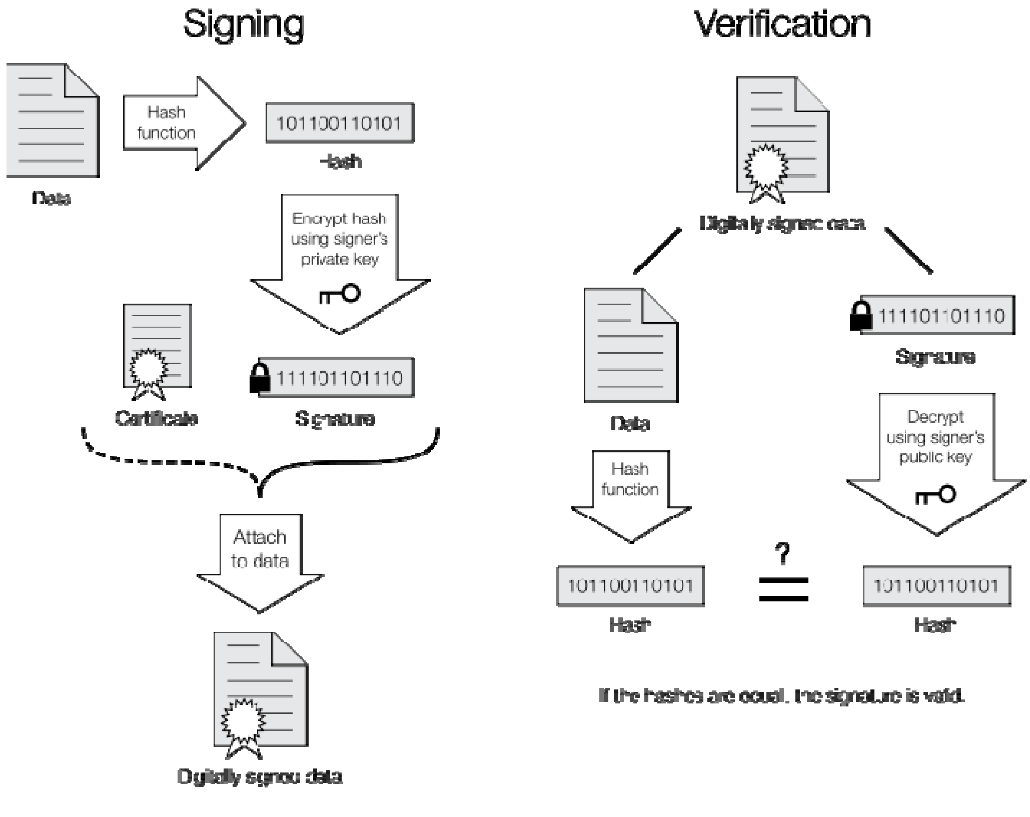


Figure 7: Digital signature generation and verification.

2.5.1 Policy based trust negotiation

Trust negotiation systems utilize digital signatures to verify the identity and other attributes of both users and services providers. In a client-server architecture mutual trust between participating entities can be built up by exchanging digital credentials before access to services is granted. However, when the credentials themselves contain sensitive information, for example credit card information and medical information, certain requirements must be meet before an entity should disclose his/her credentials. In this case a Credential Access Policy (CAP) related to every credential can be required. A CAP describes the set of preconditions that must be fulfilled by the requester of the credential before it can be disclosed [10].

TrustBuilder [24] is a research project at University of Illinois at Urbana-Champaign trying to investigate different way of building trust by using access policies and digital credentials. Figure 8 shows trust negotiation taking between a service provided by Bob (certified by a trusted third party, here assumed to be the Better Business Bureau (BBB)) and a user Alice (who possess a VISA credit/debit card). The negotiation starts when Alice request access to Bob's service. Bob replies with a policy guarding his service.

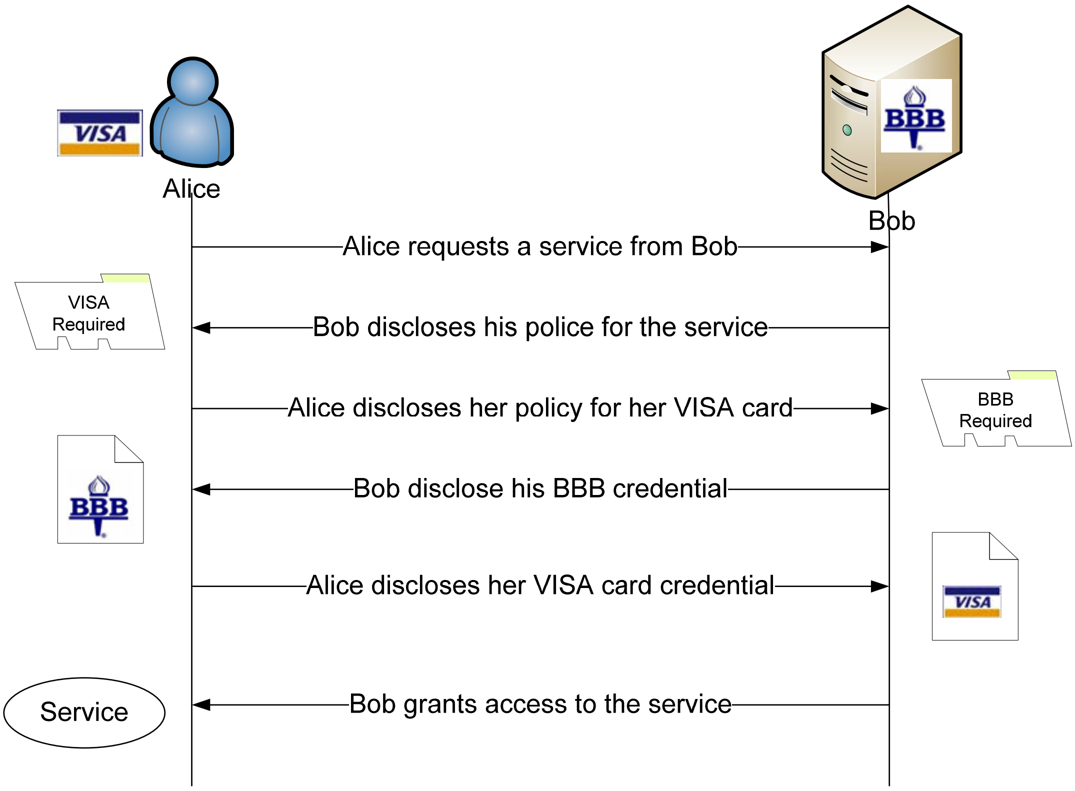

VISA
Alice
BBB
Bob
Alice requests a service from Bob
VISA Required
Bob discloses his police for the service
Alice discloses her policy for her VISA card
BBB Required
BBB
Bob disclose his BBB credential
Alice discloses her VISA card credential
VISA
Service
Bob grants access to the service

Figure 8: Policy based trust negotiation. (Adapted from [24])

Bob's service policy states that Alice must have a VISA card so that she can be billed for the service. However, Alice's CAP says that she will not disclose her VISA card credential unless she is sure that Bob's service is certified by BBB. Upon receiving Alice's CAP Bob will send his BBB credential to Alice. Once Alice has verified Bob's credential, she can send her VISA card credential knowing that Bob's service is vouched for by a trusted third party (i.e., the BBB). If Alice's credentials are found to be genuine, then Bob will grant her access to his service.

Such policy based trust negotiation models allow participants protect their sensitive credential information. The trust negotiation model presented here can easily be incorporated into existing systems because we can utilize the existing Public Key Infrastructures (PKI) functions that are already built in to various systems. For example, Minisip now supports X.509 certificates (see section 2.3.2); hence we may be able to integrate CAP into our trust negotiation model. However, more in-depth investigation need to be performed in order to understand the usability and efficiency of the model in real world scenarios.

2.6 Presence and Instant Messaging

Using presence and instant messaging enables more pleasant and effective communication compared to traditional telephone communications. In a traditional telephone system there is no convenient way of determining the status of the called party before actually making the call. If the user is not available to receive the call, the call will end up in their voice mail or may not be connected at all. In a voice over IP (VoIP) system presence information enable us to determine if the desired party is available online and is ready to take part in the communication. By using user agents that support presence information, user can set their current status and indicate their preferred contact mechanism.

SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) is an extension of the SIP protocol to support presence and instant messaging. SIMPLE provides an event based notification mechanism using SUBSCRIBE and NOTIFY messages. SIP user agents can either use an end-to-end mode where the user agents themselves handle the presence information or the presence user agents can update a presence server in a more centralized approach. In the latter case, a presence agent will receive subscription information from watchers. Then applications interested in status updates will receive notifications as the status information is updated by a presence user agent. Figure 9 shows a watcher that subscribes to a status change of the presence user agent. The presence agent will notify the watcher when status change occurs using PUBLISH messages sent from the presence user agent.

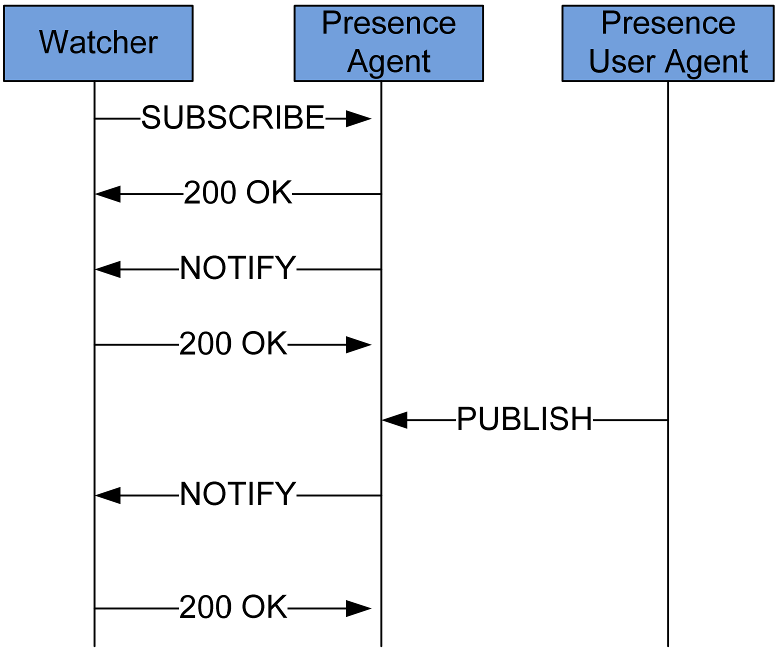


Figure 9: Message flow in a Presence System (adapted from [1]**)**

The actual presence information carried in the SIP messages is a well formatted XML document. RFC 3863 [25] defines the structure of this document in a format called Presence Information Data Format (PIDF). Important issues when using centralized presence systems are the questions of privacy and user integrity. Therefore, user agents should have some control of who can subscribe to their presence information. A standard protocol called XML

Configuration Access Protocol (XCAP) [26] is used to enable clients to manage their presence information. The XCAP protocol, implemented as a server daemon, allows user agents to access their presence information as stored in the presence server (in XML format) using HTTP. This means that a SIP user agent can set various access policies regarding their presence information. A human SIP user could do this using a web browser.

This kind of subscription based event notification architecture used in SIP is valuable when building context aware systems. Here at Wireless@KTH several master's thesis projects have exploited this architecture in building sensor systems [3] and, intelligent home and office applications [7]. We plan to utilize the lessons learned from their earlier work when designing and implementing our system. In the next chapter we summarize some of these projects done in the department and critically analyze how they can be incorporated in or leveraged by our project.

3. Related Work

3.1 Exploiting devices around us

The thesis by Shasha Zhang [5] focused on aggregating various devices connected to a local area network so that the services provided by the devices near the user appear as if they are provided by a single device. The goal is to improve the user's experience when using various devices to provide services in a personal area network. Her thesis suggests a solution based on the Service Location Protocol (SLP) (see section 2.2.3 on page 6) for automatically discovering services. The thesis tried to realize a remote desktop control service which enables small handheld devices to utilize larger screen devices around them. Devices, for example a PDA with a larger screen, in the personal area network could advertize their services by sending SLP service advertisements to an SLP directory agent running on one of the KTH and HP smart badges [9]. When a cellular phone with a small display joins the network, an IP address will be assigned by a DHCP server running on the badge[9]. An SLP user agent running on the cellular phone will discover a remote desktop service registered by the PDA and can use it to display its output. The implementation of a client application executing on the cellular phones and its integration with an SLP user agent is outlined, but no implementation or evaluation work is presented in the thesis. The author also raised some concerns related to authentication and trust in device aggregation, but no suggestion has been made of how to overcome concerns. The use of a remote display of a cellular phone's display on the user's laptop or desktop display was shown in a related bachelor's thesis project [27].

Security will be a crucial part of our thesis work and a mechanism to properly authenticate and build trust between communicating devices will be investigated and a prototype will be developed. One of the goals of our project is to study how we can effectively discover various services provided by the devices around the user. Therefore, we will build upon some of the experiments performed by Zhang‘s thesis, especially how she created SLP service agents that interoperate with the OpenSLP [28] implementation.

The idea of utilizing devices around the user for performing intensive computation is presented by S. Goyal and J. Carter in [29]. This paper describes how constrained devices can exploit surrogate machines which make their computing and networking power available. The goal is to enable applications that require a lot of computing power such as real-time voice recognition and synthetic web service application available on small handheld devices. The suggested framework uses a simple XML based querying mechanism to allow the handheld devices to locate surrogate machines. Upon discovering a surrogate, the user's device will be assigned a virtual machine in which they can run a shell script that will download and install the packages required for the user's task. The proposed system uses public key cryptography to ensure that only authorized clients will be allocated a virtual machine and to make sure that the virtual machine is used only by the client that it is created for. The idea of utilizing more

[9] The badge was used as it was equipped with both a WLAN interface and could act as a USB bus master, thus USB devices such as keyboards, headsets, etc. easily could be attached to it.

powerful devices to perform remote speech recognition and control constrained devices is also demonstrated in Johan Sverin's master thesis [30]

3.2 Occupancy sensor system

The master's thesis by Daniel Hübinette [3] presents a design for a sensor system that keeps track of the occupancy of a room. The sensor was designed to report if a room contains zero, one, or more people. Various application scenarios and possible services that can be built upon such a sensor are presented. The sensor system uses a passive infrared motion detector [31]. This sensor has the ability to determine the directional movement of a heat source, enabling a system which processes this data to determine the occupancy of a room (assuming that each doorway is only wide enough for a single person to pass through and that each doorway is equipped with a PIR sensor). The proposed architecture uses a presence user agent which sends a PUBLISH messages to update the status of the room to a context broker. An interested user agent could subscribe for status updates of a room and will receive NOTIFY messages.

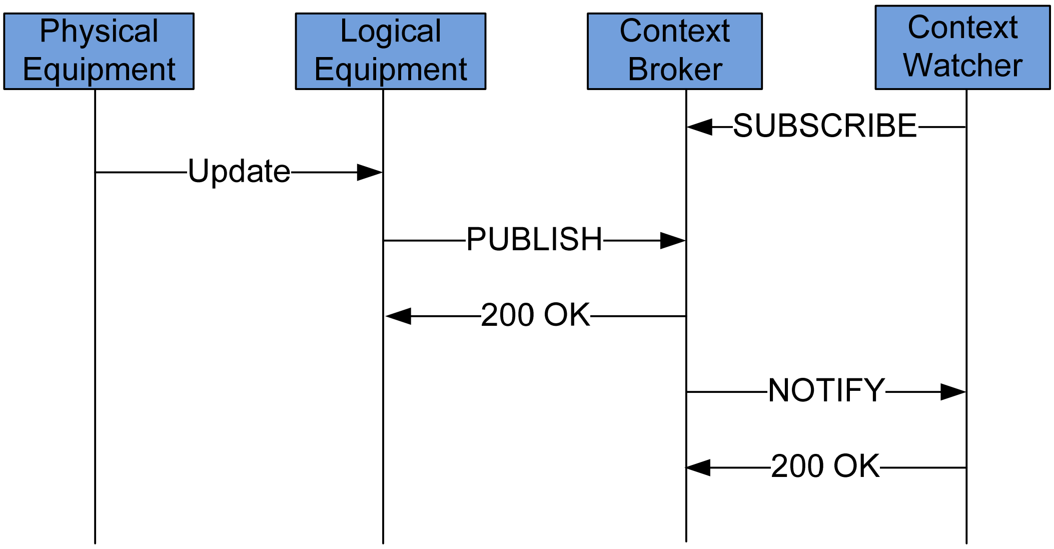


Figure 10: Event notification using context broker. (Adapted from [32]**)**

The thesis mainly focuses on the hardware design and prototyping of the sensor system. Currently the system is only able to send PUBLISH messages to the context broker and no realistic service has been developed to make use of the occupancy information gathered by the sensors. However, a prototype service was shown in the theses of Xueliang Ren [32] and Ke Wang [33].

Therefore, the focus of our thesis will be to understand how we can utilize contextual information from various sensors in order develop a secure mobile SIP user agent that makes use of available multimedia devices. Security issues, mentioned as future work in Daniel's thesis, will be central to our contributions.

3.3 Context-addressed messaging

Alisa Devlic's licentiate thesis [34] introduces the idea of utilizing the user's contextual information as the basis for addressing message. In her approach messages will be addressed not using their network specific address, but rather using contextual information of the user. By doing so users' communication will be more personal because such an approach is able to make sure that the user will only receive messages relevant in his/her current context [34]. For this purpose a context management middleware is provided. The middleware implements functionalities such as discovering new context sources, aggregating, filtering, synthesizing, and storing contextual information.

As it can be seen Figure 11, applications that use this middleware have two roles. They can consume contextual information produced by the communication and dissemination subsystem and at the same time they can serve as sensors that provide additional contextual information. In our thesis project a similar context aggregation framework is used to determine the location of the user based on information gathered from PIR sensor, IR beacons, light intensity and temperature sensors (see 4.5). However, our approach does not support a two way interaction between the application and the framework. Perhaps in the future such a distributed approach of aggregating contextual information could be incorporated in to our framework.

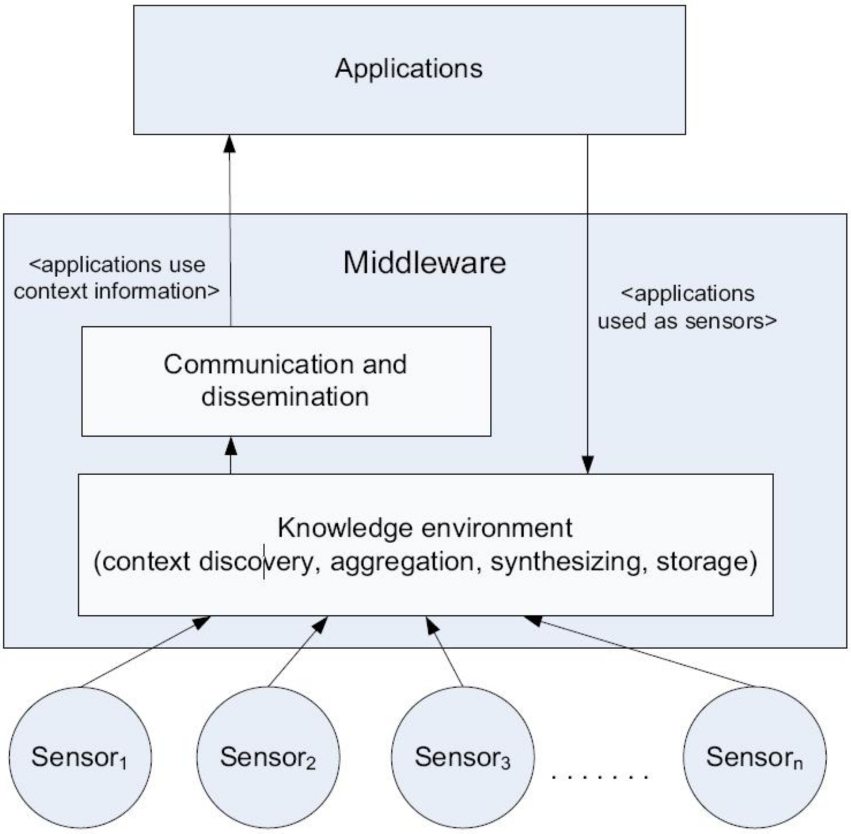

Applications
Middleware
<applications use context information>
<applications used as sensors>
Communication and dissemination
Knowledge environment (context discovery, aggregation, synthesizing, storage)
Sensor$_1$
Sensor$_2$
Sensor$_3$
. . . .
Sensor$_n$

Figure 11: Context Management Framework [34][10].

3.4 Redirecting RTP Media

Oscar Santillana master's thesis looks at different mechanisms for redirecting RTP media streams from a handheld device [4]. This thesis claims that RTP media redirection enables users with constrained computational power, a smaller display, and limited bandwidth

[10] Appears here with the permission of Alisa Devlic.

to utilize more powerful devices around them in order to provide the user with a better multimedia session. Two solutions based on third-party call control (3PCC) and the REFER method were investigated to enable seamless session mobility. The first alternative is based on a simple call flow described in RFC 3725 [35] and uses a mobile node to control the redirection of the RTP media to a different location. The user agent running on the mobile node sends a new INVITE request for every multimedia device chosen to participate in the session. The mobile node also sends a RE-INVITE request, with the updated media parameters in the SDP body, to the correspondent node. This approach is interesting because the mobile node has full control of the SIP session; enabling it to latter redirect the RTP media to the original device. Figure 12 shows this call flow.

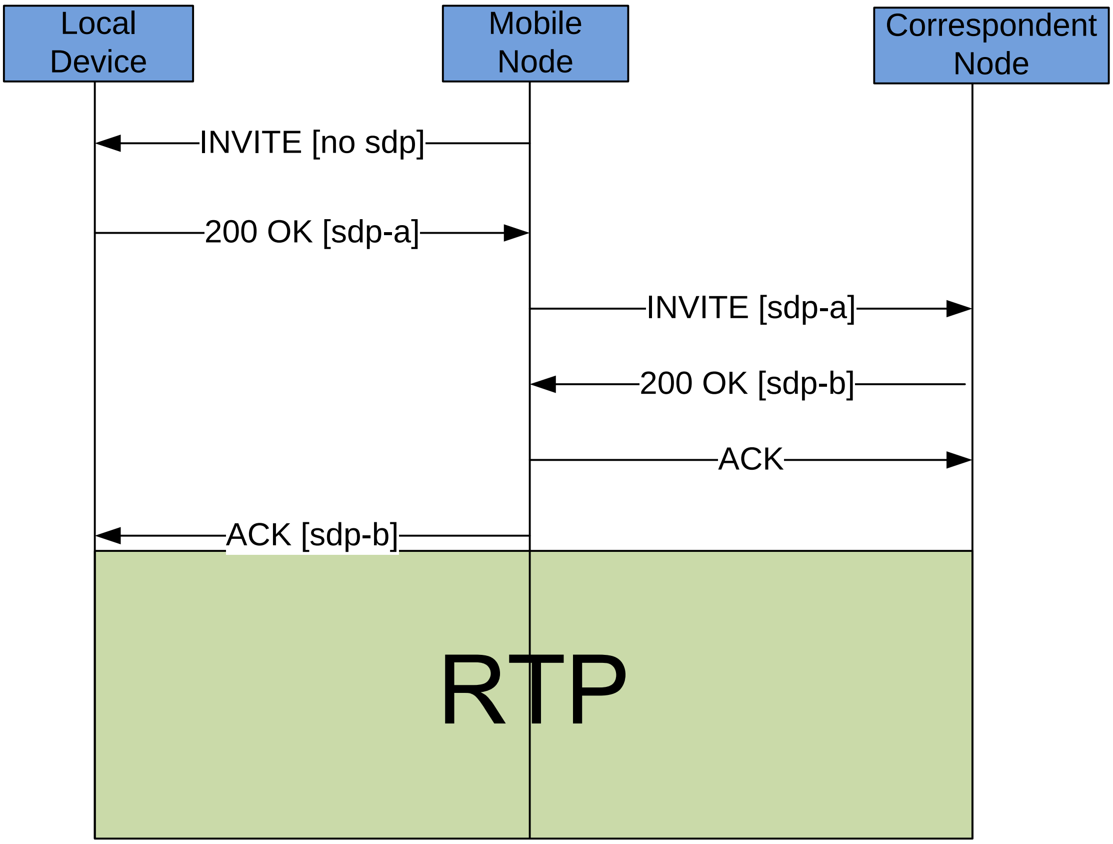


Figure 12: Third Party Call Control (3PCC) (Adapted from [4]**)**

This call flow is simple in structure, but there exists a major problem with timeouts. When the mobile node sends the RE-INVITE request, the correspondent node is expected to answer the call and send a 200 OK almost immediately. Otherwise the local device will assume that the call failed as it did not receive the ACK from the mobile node on time. In fact RFC 3725 recommends that this call flow would be more effective if the correspondent node is an entity that will ensure that the RE-INVITE will be answered almost immediately. As a result, this call flow is effective if implemented on multimedia servers, conferencing servers, or messaging servers – as they will respond quickly.

Directly using this approach for our system raises some issues. Most importantly we should study how different SIP user agents react to reception of a RE-INVITE request while being in a session. The expected behavior is that the user agent will update its media session according to the new SDP and continue the call without causing any interruption. This should be verified experimentally (see section 5.4.5 for discussion of those experiments). We will also investigate how the 3PCC call transfer affects the MIKEY key exchange. For instance, we will consider a situation where the user redirects its media to more than one local device. For each redirection the MIKEY session keys will be updated per media stream. Therefore we will study how those session keys can be securely transferred to the local devices.

3.5 An Intelligent presentation System

Hu Lidan's master's thesis aims to create an intelligent presentation system by utilizing a user's context to provide a simplified presentation experience [7]. The proposed system envisions detecting the arrival of the presenter and automatically presenting their previously uploaded slides via a projector - while allowing the presenter to control the presentation using any handheld device with a web browser. The suggested solution utilized a location sensor implemented as a presence user agent to publish the status of the user on a context-aware server. A room occupancy sensor, also implemented as a presence user agent by a separate master's thesis project [3], was used to understand if the user is going to conduct the presentation and if enough attendees are present to start the presentation. A key result of her thesis work was that her agent provides a URL to the presented that can be used beforehand to upload their presentation and can be used at the time of the presentation to control the presentation. The URL points to the presentation system (which acts as a proxy) for the user to control the actual presentation hardware.

4. Context Modeling Framework

4.1 Requirements of our application

The context model to be used in our application should answer a number of questions. The following subsections will each examine of these questions.

4.1.1 What aspects should the model include?

Space

The context model should be able to deal with the user’s location. When we say *location* we mean the relative location of the user in an home or office environment. Absolute user locations (such as world coordinates via GPS) will not be necessary for our application.

Environment

Some attributes of the user's surroundings also affect the behavior of our system. For instance, the presence or absence of various multimedia equipment around the user’s current location should be modeled in the proposed framework.

History

The model should utilize previous context history to increase the accuracy of the reasoning.

Subject

Users are central part of our context aware application. We consider context to be the user’s context (specially their location and the surrounding environment) as perceived by the application.

User’s preference

Preference information explicitly set by the user or learned though the user’s prior actions will also be used as contextual information. Users should be able to set their profile information and our system should use this information to understand the user’s choices. Example of such profile information include presence information, previous experience with multimedia devices and services, the user’s calendar, and he user's address book

4.1.2 How should we represent context?

The mechanism to be used to represent context should allow some level of automatic processing of the captured contextual data. A formal representation technique should also be available to enable reasoning support. Most importantly the mechanism should be intuitive enough to let us easily incorporate the contextual information with the application to be developed.

Furthermore, the model should be flexible enough to easily adapt to use different context information. This is important because the requirements on our system may change over time, posing a need for additional contextual information (for example, by including additional sensors) to be incorporated into the system. In such situations it is important to have a context representation mechanism which is independent of the actual context being represented and that can easily be extended.

Another important aspect related to context representation is context granularity. The framework to be developed should allow us to represent contextual information at different levels of detail depending on the application's requirement. This is particularly important because the various context sources to be used in our application (PIR detectors, light and temperature sensor, and IR beacons) provide different contextual information and with varying degrees of accuracy.

4.1.3 How should we manage context?

The context model should have a mechanism to build, manage, and utilize the context information. Context information can be gathered in either a centralized or distributed manner. In our system we employ a mixture of context sources (including PIR, IR beacon, and light and temperature sensors) to determine the user's situation. The model to be proposed should detect if the user is present in a certain room and utilize the user's previous experience and preference information to draw a conclusion about the current context of the user. The model should not be a static in the sense that all the information is predetermined at design time, but rather the model should be robust enough to accurately figure out the situation of the user at runtime.

A typical approach would utilize state of the art sensing technologies to determine the user's presence at a specific location. The sensor system to be deployed should tell us if the user entered into the room or left the room. The sensor systems proposed in [3] and [6] will be experimented with, along with other means of determining a user's presence in a room.

A robust reasoning technique should be included in the model to allow us to fine tune the context information collected from the sensors. Profile information set by the user (for example scheduled meetings, address books, and social networking knowledge base(s)) could be used to supplement the situation determination. When enough context information cannot be obtained the system should somehow interpolate and mediate the available context information to estimate the user's most likely context.

After a reasonable estimate of the user's context is determined, our application exploits this information to update the presence information of the user on a presence server. For this purpose a presence user agent capable of interworking with the context model should be implemented. The figure below illustrates the three components our model should have. The lower layer is a collection of various context sources that can provide contextual information to the system. The middle layer is the portion of the context model that analyzes the gathered context information and determines the user's situation. The output of this layer will be given to a SIP presence user agent that will update the user's situation on a presence server.

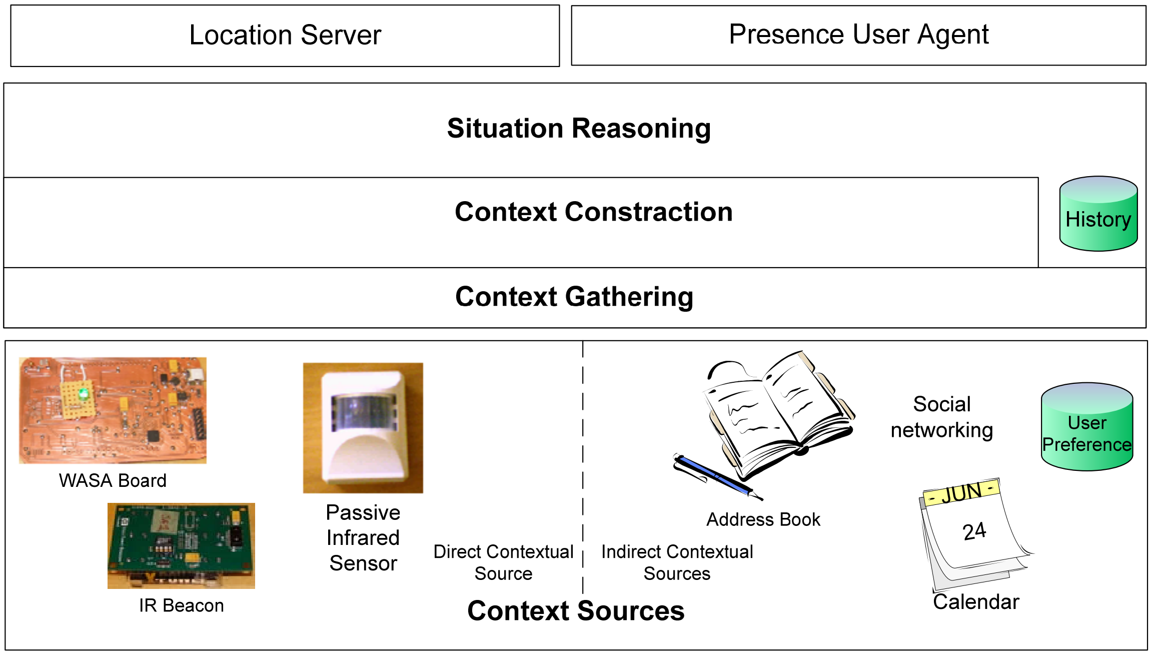


Figure 13: Components of the context model

4.1.4 Addressing privacy issues

When developing a context aware system, we must pay careful attention regarding privacy of the user. For instance, an application which uses the user's location information should utilize such information with the explicit consent of the user. Failing to do so is not only an inconvenience to the user, but it is considered as crime in many countries. In our framework the central system does ***not*** store any contextual information that can be mapped to the user's identity. Instead we propose a solution where the user has full control of which of their contextual information is exposed. Section 4.5 provides a comparison of different alternatives of building a context framework to be used in our application.

4.2 The CoolBase Platform

HP Labs created a software and hardware platform to facilitate creation of a wide verity of ubiquitous computing applications. The platform enables developers to create a personalized application that utilizes contextual information of the user. The platform leverages web presence information linked to people, places, and things [36]. An example application developed based on this platform is a virtual tour guide. It aims to deliver personalized services based on contextual information such as location and time. As the user visits each exhibit items in a museum, the user's PDA displays a web page tied to the exhibit.

The CoolBase platform consists of several software and hardware modules. Below we will describe some of the components that we have found valuable to our work.

Esquirt: This module provides an API to encapsulate a device to device interaction model. Its sole purpose is to allow devices to access the web presence of other devices by reference (for instance using a URL). For example, a mobile phone could send(or "squirt") a URL of a document to a printer which will retrieve the document and print it for the user. We have developed an Esquirt like application which identifies rooms by reference (using a unique IR beacon number obtained from a HP Cooltown IR beacon).

Web Presence Manager: This entity facilitates creation and hosting of web presence information for people, places, and things. It also provides context composing functionality. For instance in a conference room, a projector and a printer may have their own individual web presence. However, if the two devices are co-located, then the presence manger will generate a relationship between the presence information of the printer, projector, and the conference room. In our context aggregation framework, the room locator server uses a similar concept in order to aggregate contextual information from different sources and provide a more accurate location of the user (see section 4.5).

CoolTown Beacons: Beacons are small battery powered devices that emits a URL reference for a specific location (see Figure 14). For instance, a beacon near a painting could broadcast the URL for this painting, providing a web presence for this specific work of art. The receiving functionality is implemented using the Esquirt API described above. The beacons use the Infrared-Ultra protocol. The Ultra protocol is a link layer protocol which allows devices to communicate without support for device discovery, sniffing, and connection oriented services. The details of how Ultra frames are formatted can be found in [37]. In our context framework the IR beacons are used to allow the mobile SIP user agent to determine the room it is situated in.

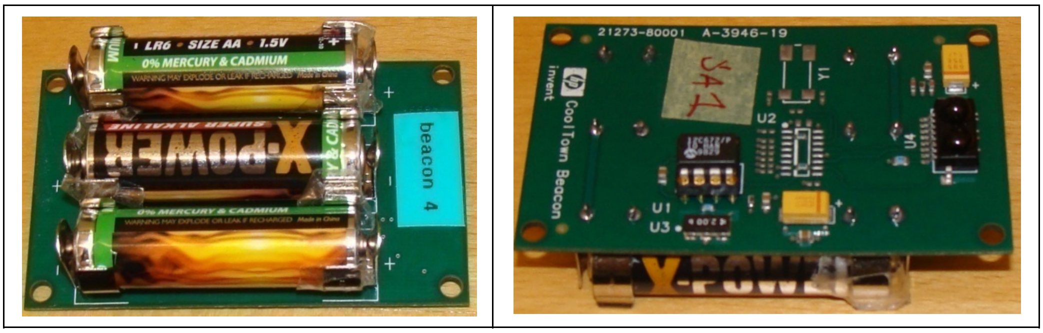


Figure 14: Cooltown IR beacon. On the left a view of the backside with battery, on the right and view of the front side with the IR transceiver.

4.3 The Wasa Sensor Board

The Wasa board is an experimental circuit board created by Mark Smith in at KTH. The Wasa board is designed to allow easy integration of sensors with an existing computing platform such us laptops and PDA [38]. The board has built in light and temperature sensors and an accelerometer which can be accessed through a USB interface. An interesting aspect of this board is that both the hardware and software designs are open, making it easy to create a customized version of the board for a specific purpose. Figure 15 shows the hardware layout of the board, while Figure 16 shows the two sides of the board.

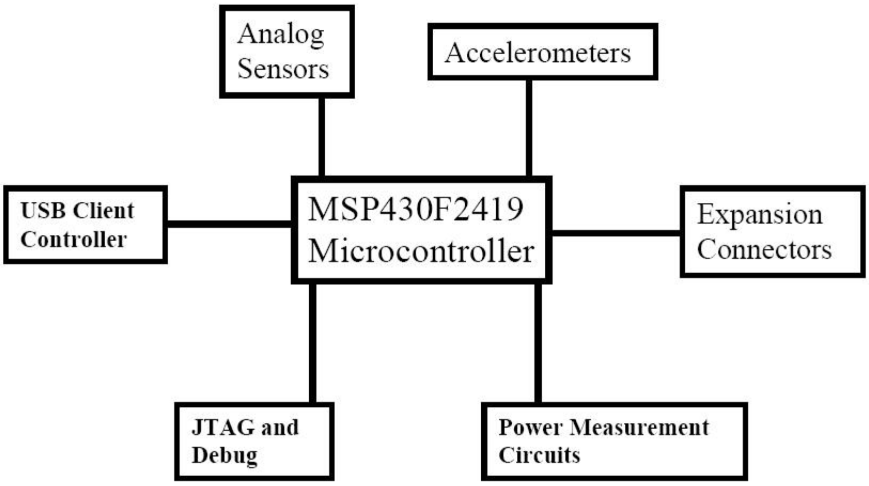


Figure 15: Block diagram of the Wasa Board [38]

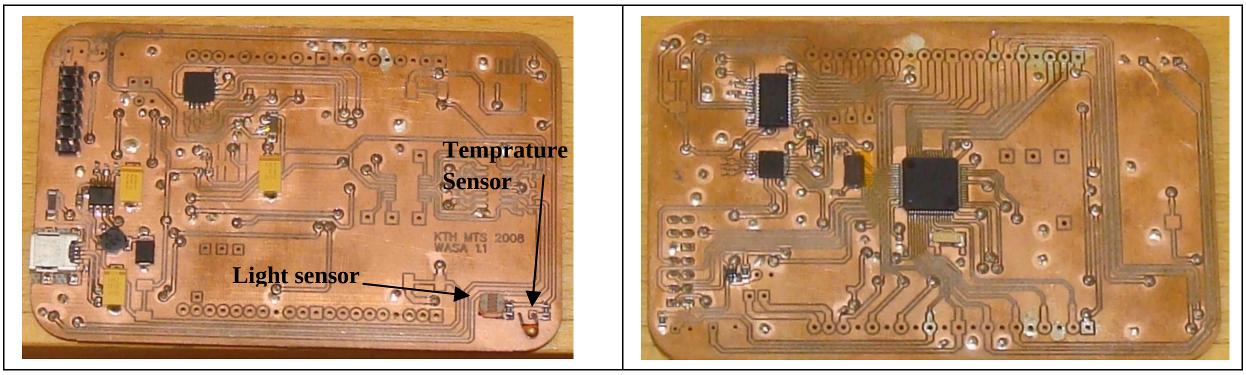


Figure 16: Wasa Board. On the left is a view of the front side and on the right a view of the back side of the board.

In our context modeling framework the Wasa board will be used to integrate light and temperature sensors. In the prototype we developed, the user's PDA includes the light and temperature readings from the Wasa board when making a location request to the room locator service. We argue that by utilizing additional sensors additional aggregated contextual information can be collected enabling us make more accurate decision regarding the location (and situation) of the user. A detailed explanation of the proposed context framework is presented in section 4.5 below and its performance and accuracy evaluation are presented in sections 6.1 and 6.2 respectively.

4.4 The Passive Infrared Sensor

In order to understand the directional movement of the user (i.e. if the user is entering or leaving a certain room), we have used a pyroelectric detector. Pyroelectric detectors are built from a special crystalline substance that takes the advantage of the polyelectric effect. The polyelectric effect is the ability for such materials to generate an electric potential when heated or cooled. For our purpose we have used the Velleman HAA52 Passive Infrared intrusion detector (see Figure 17).

Figure 17: The Velleman HAA52 PIR Detector

The most important aspect of this sensor is the ability to determine directional movement of a heat source (i.e. we can tell if someone is entering or exiting a room). This is possible because the potential difference generated as a heat source moves from left to right produces a different wave form than when someone moves past it in the opposite direction. Figure 14 shows the wave form divided in two five different zones.

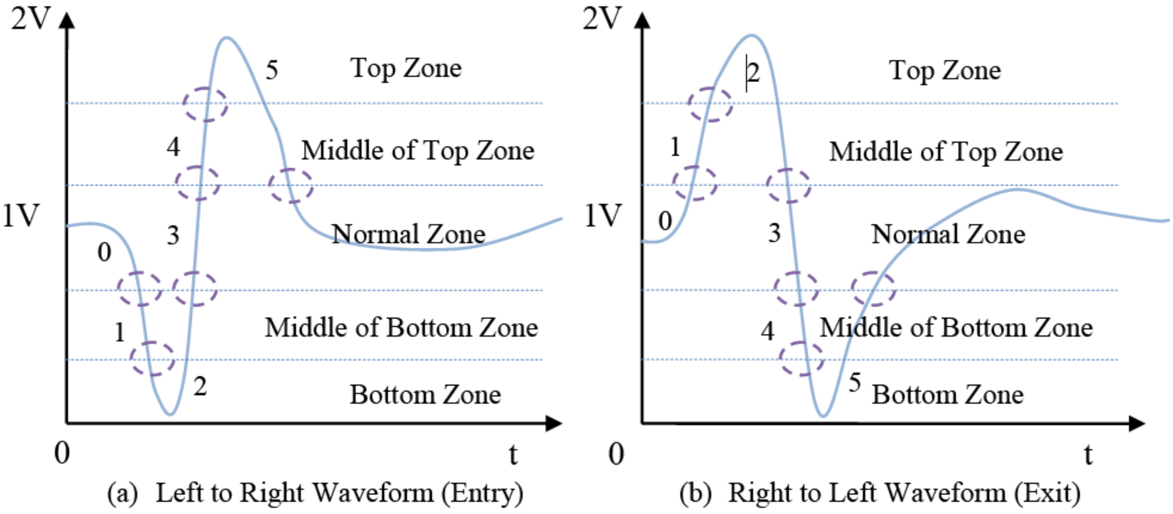


Figure 18: Different waveforms are produced depending on the directional of movement of a heat source past the sensor

Our work has benefited from previous attempts in the Wireless@KTH lab in using PIR sensors to determine directional movement of the user to develop room occupancy sensors and meeting detectors systems. Daniel Hübinette's thesis suggests using a state based algorithm which tracks changes between the five zones shown in Figure 18 to determine the directional movement of the user [3]. Recent work by Xueliang Ren showed that more accuracy can be achieved by utilizing a correlation based technique to determine the movement of the users, this approach is used in our context framework as will be presented in section 4.6.

4.5 Proposed context model

In order to understand the context of the user, we need to find a mechanism to aggregate contextual information from different sources. In this thesis the goal is to correctly estimate the location of the user in relation to a particular room -- in order to use resources located in this room. Our context aggregation framework will utilize various sensors (PIR Sensors, IR Beacons, light, and temperature sensors) to estimate the user's location[11]. With this framework in place, our application will be able to tell if the user entered or left a particular room. In this section we propose two alternative techniques to collect, aggregate, and interpreted contextual information for the user. Both alternatives allow us to aggregate context from various sources, the main difference lies in their efficiency and scalability.

4.5.1 Alternative 1: Using a mobile Presence Watcher

This alternative is based on a SIP presence watcher application implemented on the PDA. Our SIP UA will subscribe for location updates originated by the meeting detector sensor used in [3]. Such a subscription will be done based on the SIP event notification package [32]. This package uses a Presence User Agent (PUA) application that manipulates presence information for a presentee. PUAs are responsible for publishing presence information to the user's presence server. In this design alternative the Pyroelectric Infrared (PIR) sensor will be connected to a PUA that will publish a presence document when someone enters or leaves the specified room. As shown in Figure 19 the document published by the presence user agent could also include the light and temperature reading from a Wasa board mounted in this room. Because the user's device has subscribed to be notified for such an update, a NOTIFY message will be sent to each of the subscribers. It is up to the user's device to figure out its current location, then start the process of redirecting the media for the user's current session(s) to the appropriate nearby device(s).

[11] We are interested in a symbolic location of user. For instance user1 is entering the seminar room "Grimeton", user$_2$ is leaving the meeting room "Mint" and, so on.

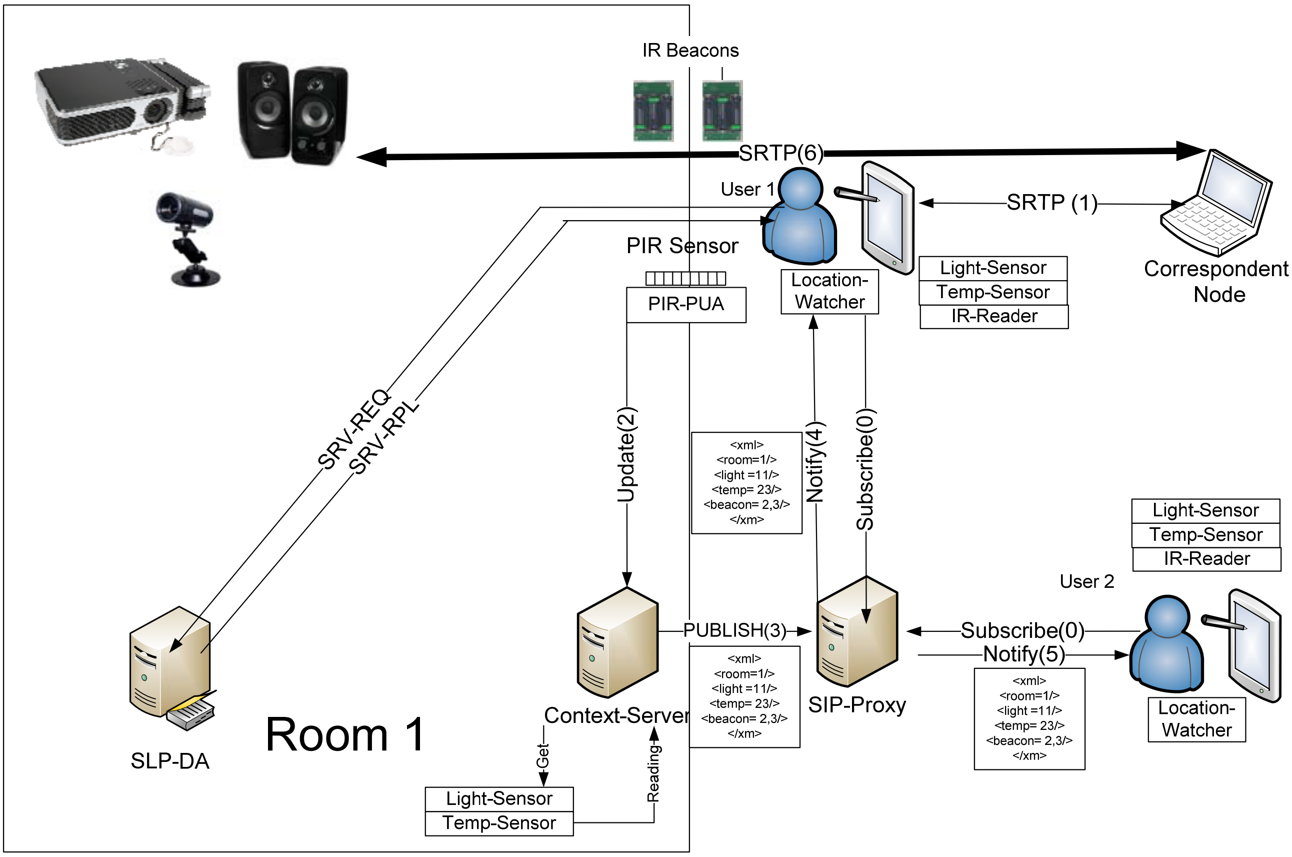


Figure 19: Mobile Watcher based Context Framework

Although most of the components for this alternative can be reused from prior work [3; 32] this approach has major scalability limitations. Consider a scenario where a handful of users has subscribed for a status update concerning room$_1$ above. When user$_1$ moves into the room the PIR presence user agent will publish a status update to the SIP proxy. This will cause a NOTIFY message to be sent to all the subscribers (as shown in Figure 19 user$_2$ will be notified even if this user did not enter room1). Because the SIP event notification package does not allow any filtering of NOTIFY messages, individual user agents will each have to act upon the notification. The mobile device can compare the IR beacon number, along with the light and temperature sensor values it has with the values that are present in the NOTIFY message to determine if the NOTIFY message is relevant to this user. However, in an environment where users enter and leave the room frequently, this approach will require a considerable amount of computing power on the mobile device. This load is proportional to the number of users entering and leaving all of the rooms that a user's agent has subscribed to for notifications.

4.5.2 Alternative 2: Using a Room Locator Service

In order to overcome the limitation of the first alternative we propose a centralized context aggregation solution. This alternative utilizes a room locator service that aggregates context information from different sources (including light sensors, temperature sensors, PIR sensors, and IR beacons) to determine the most probable location of the user. IR beacons will be installed at the entrance of every room and they will broadcast a URL specific for a given room. When the user's device is within the light of sight of the IR beacon, it will receive the URL broadcasted by the beacon; this will trigger a room location request (by the user's mobile device). Such a request will include the URL from the beacon and the temperature and light readings from the Wasa board connected to the PDA. A location request document formatted in XML will be created and will be uploaded to the room locator server using HTTP(S). Note that this approach does not require the PDA to be preconfigured with the room locator's service URL. This service URL can either be the URL from the IR beacon or it can be dynamically discovered using our mobile SLP user agent (as will be discussed in section 5.3).

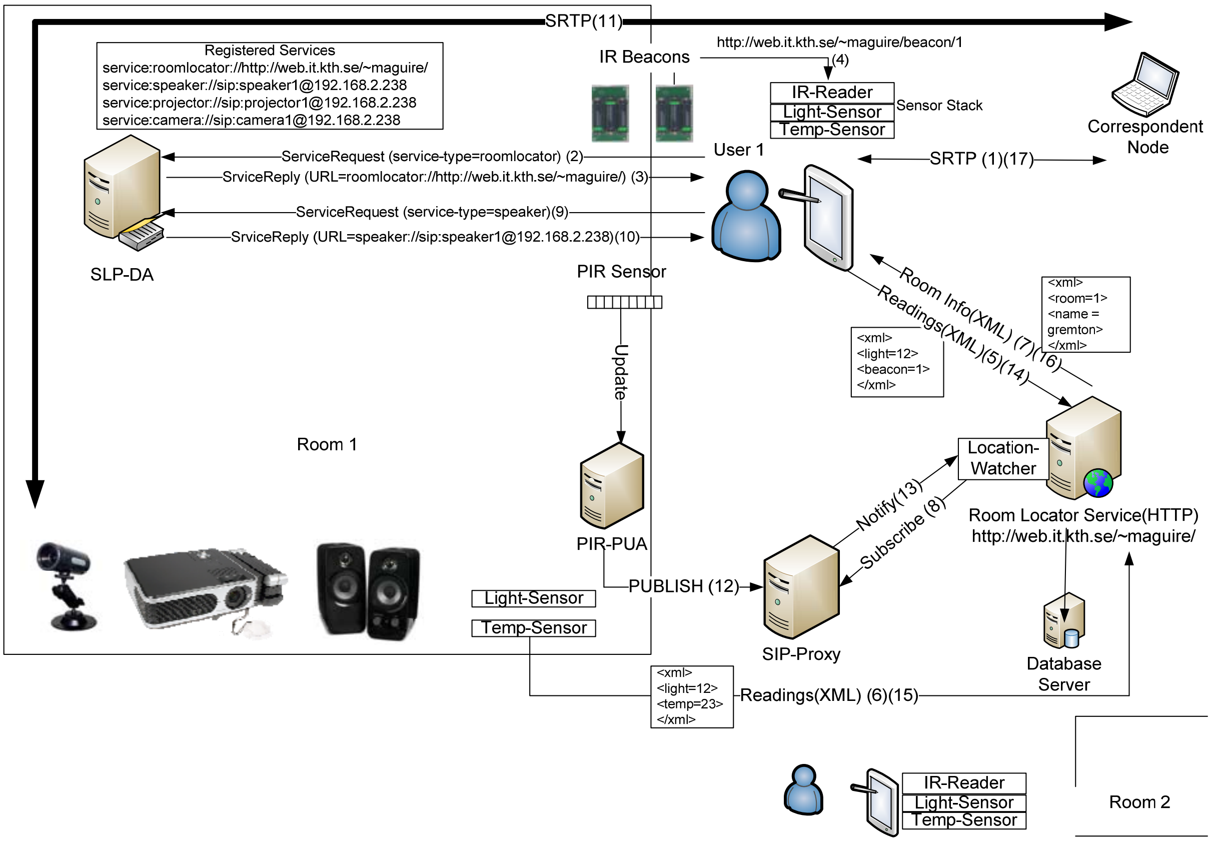

SRTP(11)
Registered Services
service:roomlocator://http://web.it.kth.se/~maguire/
service:speaker://sip:speaker1@192.168.2.238
service:projector://sip:projector1@192.168.2.238
service:camera://sip:camera1@192.168.2.238
IR Beacons
http://web.it.kth.se/~maguire/beacon/1
(4)
IR-Reader
Light-Sensor
Temp-Sensor
Sensor Stack
Correspondent Node
User 1
ServiceRequest (service-type=roomlocator) (2)
SrviceReply (URL=roomlocator://http://web.it.kth.se/~maguire/) (3)
ServiceRequest (service-type=speaker)(9)
SrviceReply (URL=speaker://sip:speaker1@192.168.2.238)(10)
SRTP (1)(17)
SLP-DA
PIR Sensor
Update
Room Info(XML) (7)(16)
Readings(XML)(5)(14)
<xml>
<room=1>
<name =
gremton>
</xml>
<xml>
<light=12>
<beacon=1>
</xml>
Room 1
Location-Watcher
Room Locator Service(HTTP)
http://web.it.kth.se/~maguire/
Notify(13)
Subscribe (8)
PIR-PUA
PUBLISH (12)
Light-Sensor
Temp-Sensor
SIP-Proxy
Database Server
<xml>
<light=12>
<temp=23>
</xml>
Readings(XML) (6)(15)
IR-Reader
Light-Sensor
Temp-Sensor
Room 2

Figure 20: Room locator service based context framework

When the room locator service gets a location request, it will execute the room locating algorithm (described in section 4.6) to determine the location of the user. The algorithm utilizes a database of IR beacon number to room mappings, along with real-time light and temperature sensor readings and status updates from the PIR sensors (to determine if the user entered or exited the room). Once the most probable location of the user is determined, a response XML document will be returned to the requester. This response will be used by the mobile device to search for multimedia devices and services available in the identified location.

This approach not only resolves the scalability problem posed by the first alternative, but also allows us to perform more accurate location determination by aggregating information from multiple sources. The locator server can also be extended to perform various reasoning and analytical operations to enable us further improve the accuracy of the whole system.

4.6 Implementation

As a proof of concept, we have implemented and evaluated the room locator service based context framework. In this sub section we will provide the details of the implementation of this framework and section 6.1 will deal with the measurements and evaluations conducted.

4.6.1 Test bed and development environment

A test bed was setup in the Wireless@KTH lab. This test bed includes two PIR sensors (PIR1 and PIR2) installed at the entrance to room 6339 and room 6340 as shown in the map below. Two IR beacons (IR-1 and IR-2) are also hanging from the ceiling above the entrance. Room 6339 is also equipped with a Wasa board (with light and temperature sensors).

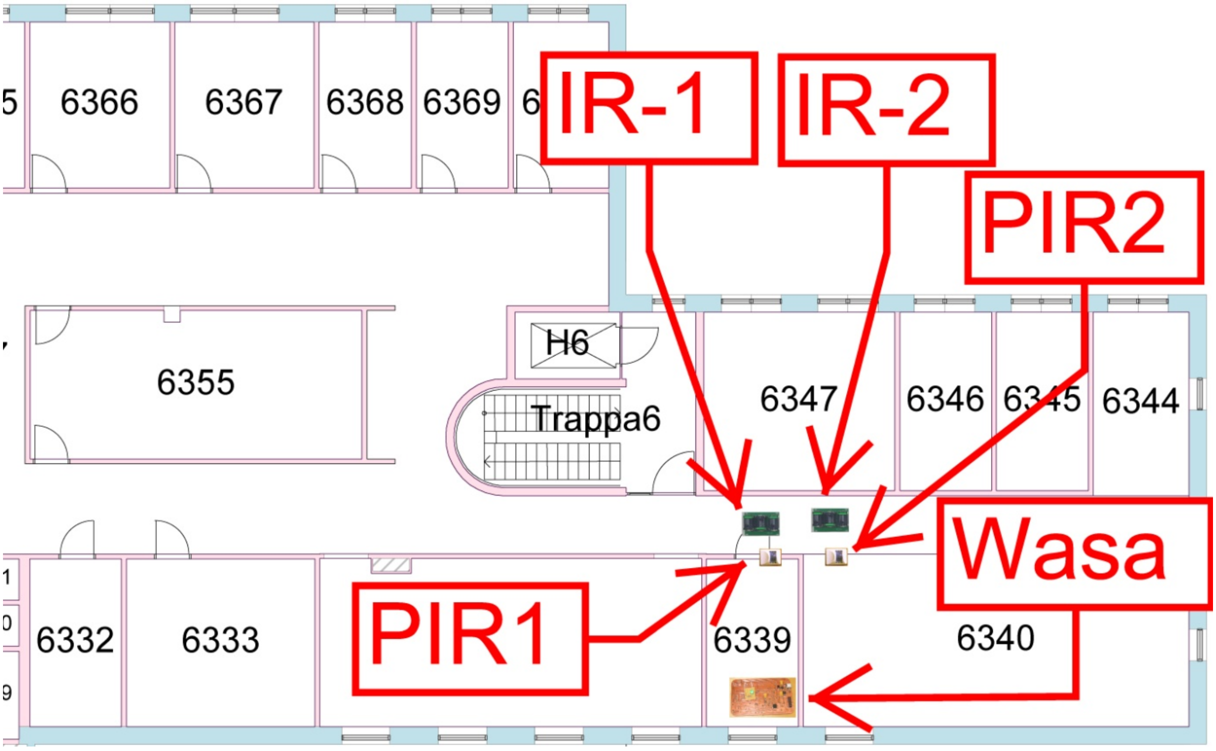


Figure 21: Installation of PIR sensors and IR beacons in the Wireless@KTH lab

Figure 22 shows the entrance to room 6340. The triangle represents the line of sight of the IR beacon and the red lines represent the optical beams used detect movement of a heat source by the PIR detector.

The two PIR sensors are connected to the Velleman K8055 USB interface board. This board allows us to access the voltage reading of the PIR sensor using a USB interface API

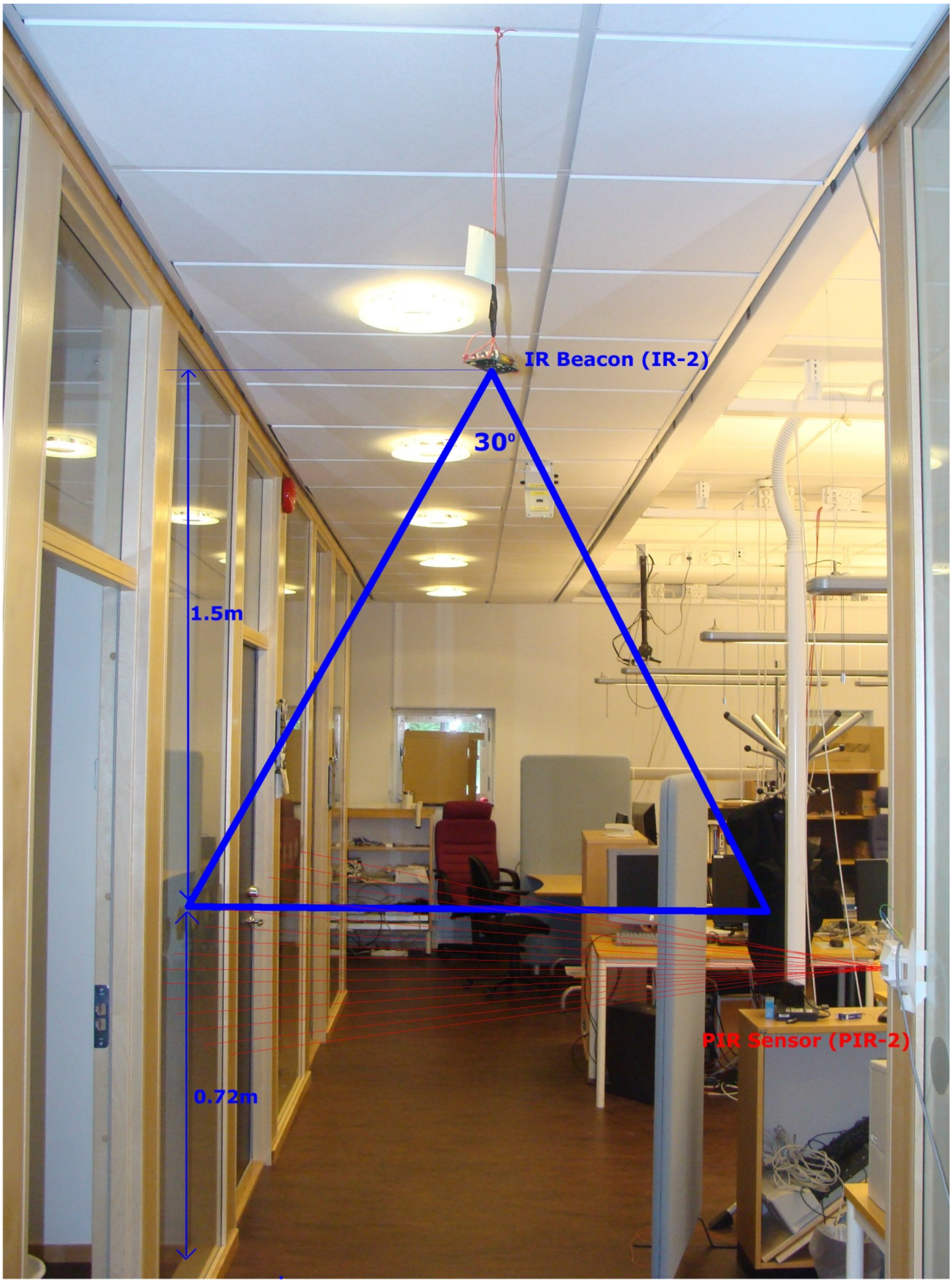


Figure 22: Snapshot of the entrance of room 6340 equipped with IR beacon and PIR sensor

that is provided with the board[12]. Because the distance between the sensors and the room locator server is greater than five meters (the maximum length of a USB cable segment, we had to use a self-powered USB hub to regenerate the USB signals. The K8055 USB interface and the Wasa board are connected to this hub, and this hub is plugged in to the room locator server as shown in Figure 23.

In order to access the ADC readings from the Wasa board we have used a Virtual Com Port (VCP) driver from FTDI Chip [39]. Appropriate drivers for both the desktop (running Windows Vista) and for the HP iPAQ PDA (running Windows Pocket PC) are available. Appendix B provides the steps required to install and configure the Wasa board on the iPAQ PDA

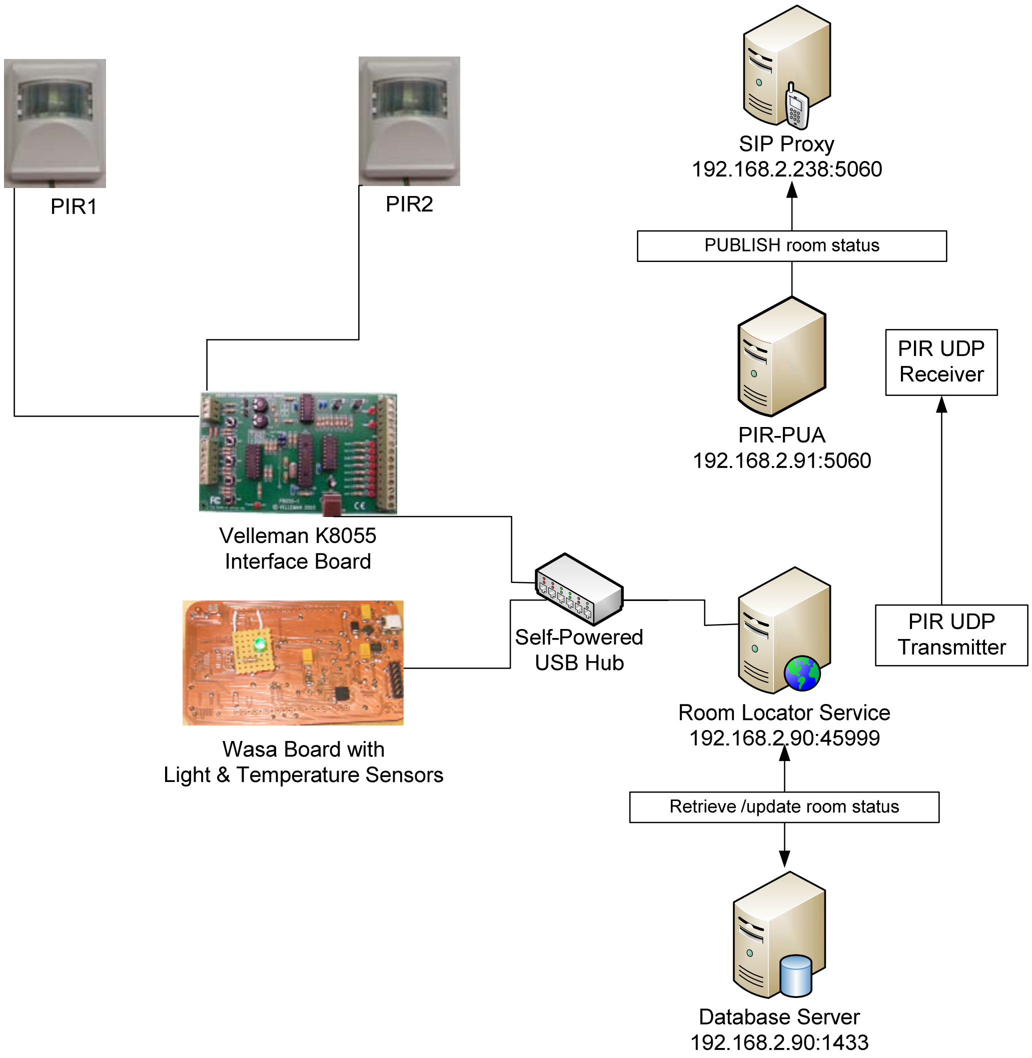


Figure 23: Components of the room locator service based context framework.

For the test bed shown above we have used three different computers (as servers) and one iPAQ PDA. The configuration details and the role played by the individual machines is summarized in Table 3.

[12] Xueliang Ren's thesis [32] provides an excellent description of this board and how to use the USB interface API.

Table 3: Configuration and role of devices used in the test bed

Machine Name	Configuration	Service Provided
CCSLEFT	Dell® OptiPlex® GX620; Intel® Pentium® Dual Core 2.4GHz; 2GB DDR2 memory; 250GB SATA-II hard drive; Dell GX62 and D-Link DGE-528T Gigabit Ethernet adapter; OpenSUSE 10.3 Linux OS	192.168.2.238:5060 – SIP Proxy, Registrar and Presence Server
CCSMOTO	Dell® OptiPlex® GX620; Intel® Pentium® Dual Core 2.4GHz; 2GB DDR2 memory; 250GB SATA-II hard drive; Dell GX62 NIC; OpenSUSE 10.3 Linux OS	192.168.2.91:5060 – Presence user agent for the PIR sensor.
		192.168.2.91:49152 – UDP receiver for PIR readings from the room locator server
CCSBEMNET	Sony® Vaio® PCG-5G2M Laptop; Intel® Core2 Duo® 1.80GHz; 2.0GB RAM; 160GB hard drive; Realtek® RTL8101 Ethernet NIC; Windows® Vista® Home Premium OS.	192.168.2.90:45999 – Room locator IIS® 7.0 Web server.
		192.168.2.90:1433 – Microsoft SQL Server 2005.
		COM16: - Reads the light and temperature ADC values from the Wasa board using VCP driver from FTDI Chip [39]
		K8055 Board Address 0: - Reads the voltage values from the PIR sensor using the K8055 USB interface board.
		192.168.2.90:ANY – Sends PIR readings as UDP packets to the PIR user agents (i.e. CCSMOTO).
CCSIPAQ	HP iPAQ H5550; Intel® XScale 400 MHz; 128 MB SDRAM; 48 MB ROM; 802.11b WLAN; IrDA up to 115.2 Kbps; photo-sensor for automatic backlight adjustment; Microsoft® Pocket PC 2003 OS	192.168.2.99:5060 – Our SIP user agent.
		192.168.2.99:ANY – Room locator client application.
		COM0: – Reads the light and temperature ADC values from the Wasa board using VCP driver from FTDI Chip [39]
		COM2: – Reads the IrDA port of the PDA using a serial port API.

4.6.2 IR-Reader for the iPAQ PDA

As shown in the test bed, the rooms containing a multimedia device have IR beacons that broadcast a unique URL using the Ir-Ultra protocol [37]. The user's PDA will read this URL to initiate a room location request. The HP iPAQ PDA has a built in IrDA port that can be accessed by a program. The IrDA port is accessed using a virtual serial port, whose address is specified by the *Index* sub key of the registry record *HKEY_LOCAL_MACHINE\Drivers\Builtin\IrCOMM*. We have used the Remote Registry Editor application provided with Visual Studio to access the registry record and we have found that it uses *COM2* by default. In order to open this port from our application we have to first disable the IR beam functionality via the Pocket PC operating system (so that other applications do not use it). This can be done by unchecking the box marked Receive all incoming beams under Setting>Connections>Beam menu.

We have developed a simple C#.Net smart device application to read the Ultra frames broadcasted by the IR beacons. Relative to the CoolBase platform described in section 4.2, this application only implements the receiving portion of the ESquirt API. The .Net Compact Framework 2.0 *System.IO.Ports* namespace provides an event based serial port access API for Pocket PC. Although the API works flawlessly for accessing standard serial ports, it does not work properly for emulated IrDA ports. This issue is discussed in the MSDN forum [40] and it is suggested to use an open source serial port API from OpenNetCF [41]. Therefore, we have used this OpenNetCF serial port API.

The *IRBeaconReader* class presented in Appendix C provides a partial listing of the code used to implement this functionality. When the instance of this class is created the constructor will create a serial port and configure it as port address = COM2, baud rate = 9600, byte size = 8, no parity bit, and stop bit = 1. When the *startReading()* method is called, the *DataReceived* and *OnError* event handlers are initialized and the *Open()* method will be called to start receiving data. When data arrives via the port, the method delegated to handle this event will be called (i.e., *OnData()* method) and the data will be retrieved from the input buffer and the appropriate parsing will be done to extract the URL and the beacon number.

We have noted that it takes two read operation to extract the full URL from the IR beacon and implemented the *OnData()* method accordingly. When this code is called for the first time it returns an ASCII encoded string containing “???????????pphttp://www.it.kt”. When called for the second time a string similar to "h.se/~maguire/beacon/1\0??" is read. The URL extracted from this two strings is the address of the room locator server (i.e., http://www.it.kth.se/~maguire/beacon/1[13]), where the number following the *beacon* directory will be used as the unique identifier for the beacon. The beacon number along with the temperature and light sensor readings will be used to create a location request XML file that will be uploaded to the room location request server. The next section presents the implementation of the temperature and light sensors used in the project.

4.6.3 Light and temperature sensors

In order to read the light and temperature values, we have connected the iPAQ PDA to a Wasa board. The Wasa board uses an FT232R USB UART interface chip from Future Technology Devices International Ltd (FTDI). Virtual Com Port (VCP) USB drivers for

[13] Actually this URL points to a subdirectory of the personal website of Professor Gerald Q. Maguire Jr.. We have used a simple PHP script at his site to redirect all HTTP requests to this URL to our location server, i.e to the URL http://192.168.2.90:45999

various platforms including Linux, Windows, and Pocket PC are available from FTDI's website [39]. We have used the iPAQ extension pack to install a USB host interface on the PDA. Appendix B provides detailed instruction on how to install and configure the Wasa board on the iPAQ. Once the board has been installed and configured properly, the light and temperature readings can be accessed using a virtual serial port.

For this purpose we have developed a C#.Net application that uses the *System.IO.Ports.SerialPort* class provided with the .Net Compact Framework to access the Wasa board. A partial listing of the *WasaBoardReader* class is provided in Appendix D. When creating an instance of this class, a serial port connection will be established with the following settings. Port = COM0, baud rate = 115200, no parity bit, byte size = 8, stop bit =1. When the *readValue()* method is called, the appropriate analog input will be read depending on the requested sensor type.

The Wasa board uses AT commands to control the devices mounted on the board. AT commands are widely used to configure MODEMs, mobile phones, and other devices using a simple terminal emulator or serial port API. The Wasa board supports three different kinds of AT commands.

Basic AT Commands: These are standard commands that are used to set up the terminal interface. We have used these commands to configure the serial port before starting to read sensor data. Table 3 lists three of the basic AT commands currently supported by the Wasa board.

Table 4: Basic AT commands supported by the Wasa board

Commands	Description	Example
V Command	Used to turn **verbose** mode on or off. If on , then each command will return OK or ERROR.	AT V1<CR> AT V0<CR>
E Command	Used to turn the command line **echo** on or off	AT E1<CR> AT E0<CR>
Q Command	(i.e., **Quiet** mode) used to set if any response to command line is sent to the terminal.	AT Q1<CR> AT Q0<CR>

S-Register Commands: These commands are used to configure, read, and write individual signal lines. Currently the Wasa board supports reading the analog inputs. From the schematic of the Wasa board [38], we have noted that the light and the temperature sensors are connected to analog inputs 5 and 6 respectively. Therefore the S-Register AT command to retrieve the readings from the light and temperature sensors can be issued as follows.

```
AT S206?<CR>  //reads the ADC value of the light sensor and
AT S205?<CR>  //reads the ADC value of the temperature sensor
```

Extended Commands: These commands are used access special devices on the Wasa board. Currently the only device that implements the extended commands is the 3D accelerometer. For instance the command `AT+OAx` reads the vector of X, Y, and Z values of the 3D accelerometer. Using information from this accelerometer we can infer than the user is moving when the device is accelerated in some direction(s).

The *readValue()* method of the *WasaBoardClass* issues the S-Register command. After issuing the command the method performs the appropriate parsing of the data available in the input buffer of the serial port. The light and temperature sensors are connected to a 12bit ADC, which means that the value will range from 0 to 4095. The method returns the mean of the consecutive read operations requested by the caller. In our case we have used five consecutive readings.

The next step will be to convert the raw ADC readings in to actual light intensity and temperature units. One option here can be to use the raw ADC reading to create the location request XML file, upload it to the locator service, and compare it with the raw ADC readings obtained on the server side. However, this option poses a major interoperability limitation on our context framework. In the ideal situation we would like our context framework to work with different light and temperature sensor types on the user device and the locator service. Although this approach could work with the current setup we have (since we use the Wasa board both on the PDA and on the server), we have decided to convert the raw ADC reading in to actual temperature and light intensity unites before creating the location request XML file.

4.6.3.1 Computing the light intensity value

The human eye can detect an electromagnetic radiation with wavelengths between 380nm and 760nm. Wavelengths longer than 760nm (infrared) and shorter than 380nm (ultraviolet) are also considered as light, but are invisible to the human eye. In order to measure light intensity, photoresistors (also known as a light dependent resistor - LDR) can be used. They are made up of a high resistance semiconductor substance (such as Cadmium Sulfide - CdS), whose resistance drops when a light source with strong enough energy in the frequency to which it is sensitive falls on its surface. Depending on the type of light sources, that is to be measured, there are different photoresistors can be used. The Wasa board uses the FUJI & CO. (Piezo Science) MPY-20C48 LDR [42]. It has a peak spectral sensitivity within the range 540-570nm, which also represents the most visibly wavelength range for human eye. The spectral sensitivity curve in Figure 24 compares the wavelength sensitivity of the MPY LDR and that of the human eye.

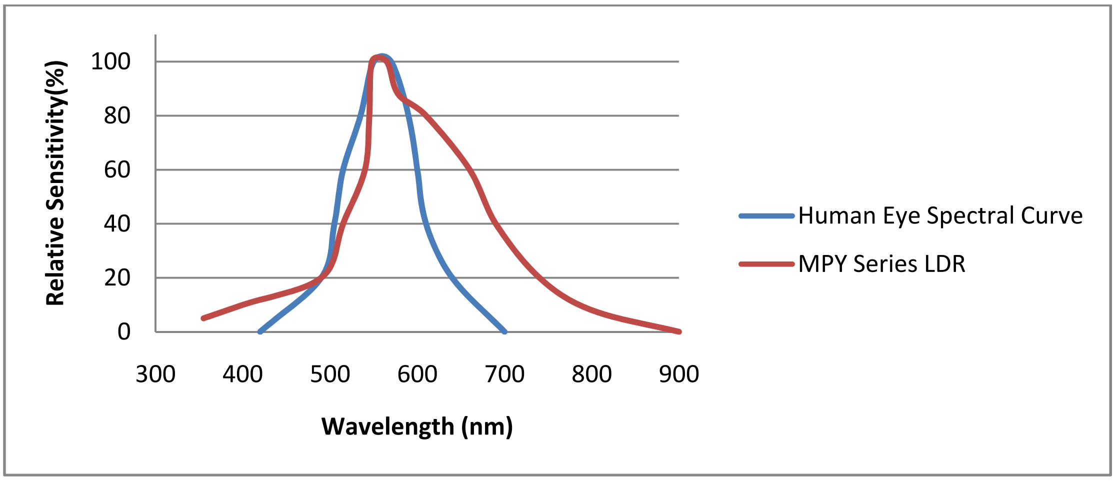


Figure 24: Comparison of peak wavelength sensitivity for the MPY series LDR and human eye. (Curves drawn by the author based on the spectral curve found at http://www.fuji-piezo.com/photoldr.htm)

Typically light intensity is measured using a standard unit called *lux*. One *lux* is equivalent to one *lumen* per square meter. The difference between *lux* and lumen is that, *lux* takes in to account the area in which the luminous flux is spread. Our aim here is to calculate the light intensity of the room based on the reading we obtain from the WASA board's ADC. As mentioned earlier the resistance of the photoresistors varies depending on the intensity of the light source it is exposed to. Figure 25 shows the relationship between luminance (in *lux*) and resistance (in KΩ) for the MPY series LDRs.

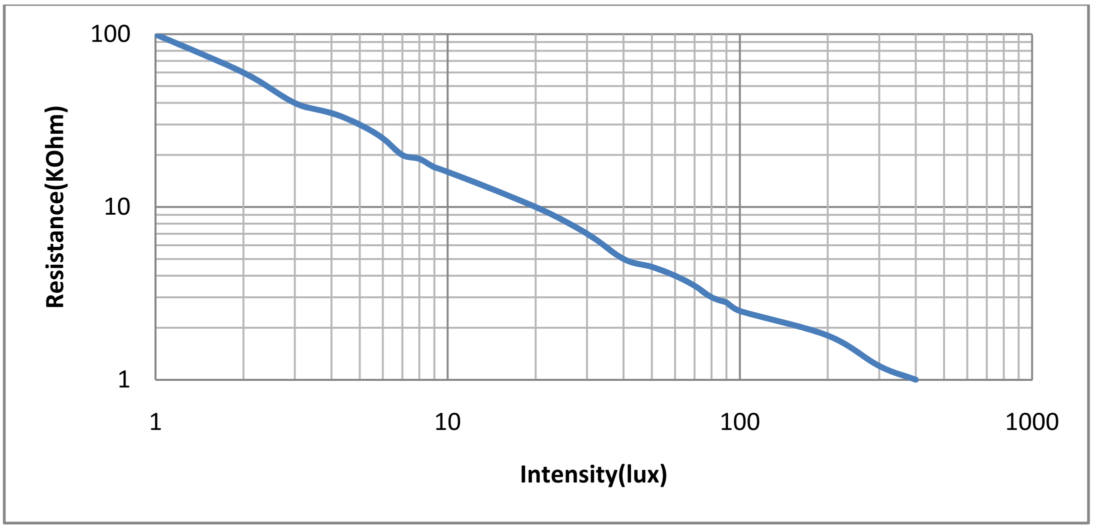


Figure 25: Relationship between luminance and resistance for the MPY LDR used on the Wasa board [42] (Curves drawn by the author based on the spectral curve found at http://www.fuji-piezo.com/photoldr.htm)

It is known that the resistance of photoresistors decreases with the light intensity in a non-linear fashion, which is close to the power law [43] (see Equation 1).

$$\frac{I}{Io} = \left(\frac{R}{Ro}\right)^{-gamma}$$

Equation 1: Power relationship between resistance and intensity.

Where *Io* is intensity at resistance *Ro*. For our purpose we have considered the reference point *Io* = 1 lux and *Ro* = 100KΩ (see Figure 25). The constant *gamma* has a value that ranges between 0.6~0.8. The exact value can be found on the datasheet of the device[14]. For the MPY-20C48 LDR used on the Wasa board *gamma* has the value 0.8.

With this information, we can now calculate the relationship betwen resistance value *R* and intensity *I* using Equation 1. In order to obtain the resistance value of the LDR from the ADC reading, we have to consider the LDR circuitry of the Wasa board (see Figure 26).

[14] The LDR used on the Wasa board has a part number MPY-20C48.

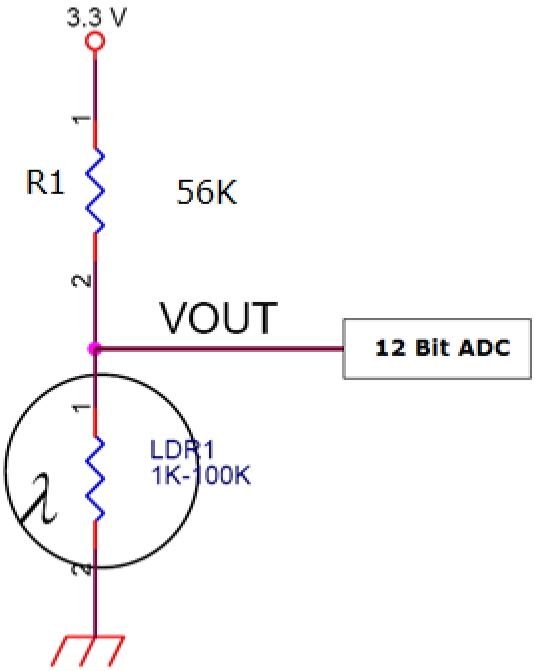


Figure 26: LDR circuit for the Wasa board

Because we have a 12 bit ADC, when we issue the appropriate AT command the ADC returns a number which ranges between 0 and 4095 (2^{12}-1). This number depends on the value of *Vout*, which is the voltage across the LDR as shown in Figure 26. In the extreme case where the ADC reading is zero, *Vout* = 0V and when it is 4095 it means *Vout* = 3.3V. Therefore, for a given ADC reading X_{ADC}, *Vout* is given as:

$$Vout = \frac{X_{ADC} \times 3.3V}{4095},$$

Equation 2: Conversion of the ADC reading to voltage

The current that passes through the LDR, I, is calculated using *Ohms law*. Note that we ignore the current that flows to the ADC.

$$I = \frac{V_{R_1}}{R_1} = \frac{3.3V - Vout}{R_1},$$

Equation 3: The current that pass thought the LDR

Finally the resistance of the LDR, R_L, is obtained using *Vout* and *I* as per *Ohms law* as follows:

$$R_L = \frac{Vout}{I},$$

Equation 4: Resistance of the LDR

Once we obtain the resistance of the LDR, then the corresponding light intensity value can be computed using Equation 1. This light intensity value in units of lux will be used when creating the location request XML file to be uploaded for the locator service. The next section will present how a similar transformation from ADC reading to actual temperature unit is performed.

4.6.3.2 Computing the temperature

The Wasa board uses a thermistor, a special kind of resistor whose resistance varies with temperature. Besides temperature sensors, thermistors are used in self-resetting circuits, inrush current limiters, and self-regulating heating elements. Compared to Resistance Temperature Detectors (RTD), thermistors are able to achieve a higher precision over a limited temperature range. This make thermistors suitable for our application, since the temperature variance within a building in a home and office environment is not significant enough to be detected by RTDs.

The relation between resistance (in Ohms) of the thermistor and the surrounding temperature (in Kelvin) is specified using a third-order approximation called the *Steinhart-Hart* equation:

$$T(R) = (a + b\, ln(R) + c\, \ln^3(\mathrm{R}) + \ d\, \ln^4(\mathrm{R}))^{-1}$$

Equation 5: Steinhart-Hart equation

Where a, b, and c are called Steinhart parameters and are provided by the manufacturer of the device. Based upon the datasheet of the NTC 2322-640-63103 thermistors used on the Wasa board[15], theses the parameters are:

$$a = 3.354016 \text{ X } 10^{-3}$$

$$b = 3.49502 \text{ X } 10^{-4}\ \mathrm{K}^{-1}$$

$$c = 2.095959 \text{ X } 10^{-6}\ \mathrm{K}^{-2}$$

$$d = \ 4.260615 \text{ X } 10^{-7}\ \mathrm{K}^{-3}$$

The resistance of the thermistor is obtained from the ADC reading in a similar approach to that of the light sensor as described in section 4.6.3.1. Now the temperature reading obtained using Equation 5 can be included when creating the location request XML file. Note that the equation only needs to be solved once for each of the 212 values that the ADC can measure, thus at run time converting an ADC reading to a temperature (or light level) is simply a table lookup.

4.6.4 Room status watcher

The watcher application is a simple SIP Presence watcher application that subscribes for status updates of various rooms. The status updates of these rooms are collected using the PIR detector and published to a SIP proxy server using the PIR user agent implemented by Xueliang Ren [32]. The main purpose of this watcher application is to provide additional information (someone entered or left a specific room) to the room locator service. Due to the scalability limitation of the mobile watcher based context framework discussed in section 4.5.1, we have decided to implement the watcher functionality on a desktop machine. The watcher will subscribe for status updates of the different rooms by sending the appropriate SUBSCRIBE message to the SIP proxy. When someone enters or leaves the room, the PIR user agent will send a PUBLISH message to the SIP proxy. As a result the SIP

[15] The termistor used on the Wasa board is from Vishay BC Components (negative temperature coefficient thermistor – NTC) has product number of 2322-640-63103. . The datasheet is available at: http://parts.digikey.com/1/parts/953698-thermistor-10k-ohm-ntc-leaded-2322-640-63103.html

proxy will send a NOTIFY message to our room status watcher application which will update this information on a database server. Finally the room locator service will use this information when making a decision about where the user is.

Xueliang Ren's thesis [32] provides an implementation of the PIR user agent to provide contextual information related to meetings going on in a room to a SIP proxy server. We have used his user agent with some extensions. The primary extension was to extend the structure of the data published by the user agent. Because we are interested in identifying if someone entered or exited a given room, we have added an *action* attribute to the XML PIDF content published by the user agent. Below we can see the new XML PIDF file published by the user agent when someone *enters* into the meeting room named "MINT"

```
<?xml version="1.0" encoding="UTF-8"?>
      <presence
xmlns="urn:ietf:params:xml:ns:pidf"entity="sip:pir@130.237.15.238">
            <tuple id="6sJ8J0">
                  <status>
                        <basic>open</basic>
                        <area>Mint</area>
                        <occupancy>Meeting</occupancy>
                        <action>entry</action>
                  </status>
                  <note>2</note>
                  <contact priority="0.8">ccsmoto</contact>
            </tuple>
      </presence>
```

Listing 1: XML PIDF published by the PIR presence user agent

The watcher application is implemented using an open source Java SIP stack called JAIN-SIP [44]. JAIN-SIP is chosen for its simplicity and comprehensive documentation. Appendix E provides partial listing of the *RoomSubscriber* class that implements this functionality. The first part of the *init()* method configures and initializes the *SipStack*. This includes setting up IP listening point, transport protocol, initializing *SipProvider*, and adding a listener for SIP events. Once this is done successfully, the second part of the method will create a SUBSCRIBE request and send it to the SIP proxy. JAIN-SIP uses an event based approach for handling SIP requests and responses. The *RoomSubscriber* class implements the *processRequest()* and *processResponse()* methods of the *SipListener* interface to handle requests and responses from a remote peer respectively.

When the proxy server sends the NOTIFY request, the *processRequest()* method will be called. The implantation will extract the body of the request and use it to update the room database. For this purpose we have created a room database table as shown in Table 5.

Table 5: The room database table

id	Name	IR beacon	Last entrance	Last exit
1	OpenArea	3	4/24/2009 14:38:54	4/24/2009 11:28:56
2	MINT	1	4/24/2009 11:51:20	4/24/2009 14:30:44

Table 6: The room history database table

Id	Room id	IR beacon	Last entrance	Last exit	Device light	Device Temp	Room light	Room temp	Action PIR	Action light	Action temp	Action final
1	1	3	4/24/2009 14:38:54	4/24/2009 11:28:56	309	200	320	210	Entrance	Entrance	Exit	Entrance
2	2	1	4/24/2009 11:51:20	4/24/2009 14:30:44	408	300	400	200	Exit	Exit	Entrance	Exit

The *updatePIRState()* method uses Java Database Connectivity (JDBC) to establish a connection to our SQL Server [16] and update the *last entrance* and *last exit* fields the room table. This update is done depending on the action (i.e. entry or exit) published by the PIR presence user agent. The *last entrance* field will be updated to the current system time if the NOTIFY message indicates that this is an entry action. Otherwise the *last exit* field will be updated to reflect an exit action. This information will be relevant for the room locator server to determine the location of the user.

4.6.5 Room locator client

The room location client is a simple web client application that creates the location request XML file (containing the IR beacon number, light and temperature readings) and uploads it to the locator server. The locator client is implemented with the SIP user agent and it is implemented using two methods defined in *IRBeaconReader* class (see Appendix F). The *createRequest()* method uses the *WasaBoardReader* class we described in section 4.6.3 to compute the light and temperature values on the user's PDA. The request will also include the IR beacon number read by the user's PDA. Finally the method will return a string representing this request. Listing 2 shows an example location request XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="request" id="22524141dfdad2472af8d1f7"
time_stamp="225451787785">
    <beacon>4</beacon>
    <light>345</light>
    <temp>223</temp>
</message>
```

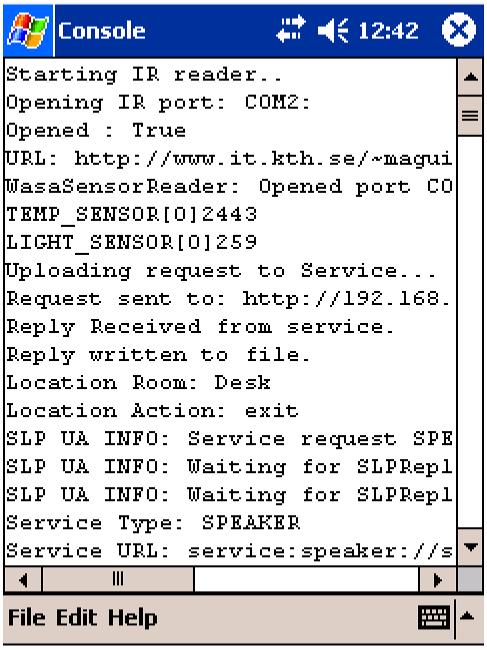
Listing 2: Example of a location request XML

[16] We have used Microsoft SQL Server 2005 Express Edition which is freely available at http://www.microsoft.com/Sqlserver/2005/en/us/express.aspx

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="reply" id="22524141dfdad241f7" time_stamp=225451787785>
    <room id="3" name="Desk" action= "exit" certainty="0.90" />
</message>
```

Listing 3: Example of location response XML

The *uploadRequest()* method will take this XML file and upload it to the room locator service. The *HttpWebRequest* class provides the appropriate methods to create an HTTP connection stream to the room locator service and to write the XML file to this stream. The file will be encoded in ASCII character encoding and will be sent as the body of a POST HTTP request. After the request is uploaded, the *GetResponse()* method of *HttpWebRequest* is used to get a handle to the input stream to read the response from the locator service. The response of the location request is also formatted in XML (see Listing 3). The *uploadRequest()* method concludes by parsing the XML reply and creating and returning an instance of the *LocationReply* class to be used for searching for multimedia services in the identified location. Appendix F provides the code snippet for this method that implements this functionality. The screenshot in Figure 27 shows the room locator client application that Listing 3 creates a new XML location request and uploads a request to the server's URL. The reply is subsequently used to search for available services using SLP (in the case shown in Figure 27 we find a speaker service).



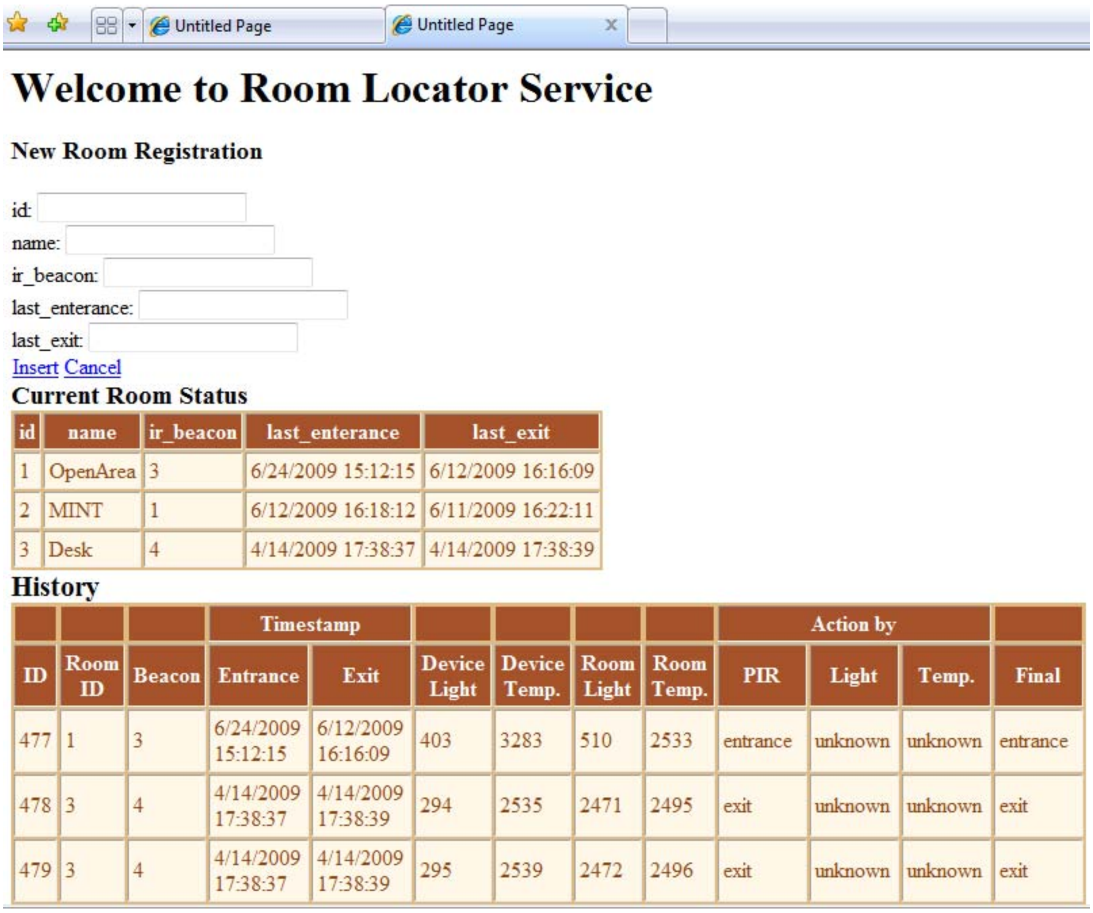
```
Starting IR reader..
Opening IR port: COM2:
Opened : True
URL: http://www.it.kth.se/~magui
WasaSensorReader: Opened port CO
TEMP_SENSOR[0]2443
LIGHT_SENSOR[0]259
Uploading request to Service...
Request sent to: http://192.168.
Reply Received from service.
Reply written to file.
Location Room: Desk
Location Action: exit
SLP UA INFO: Service request SPE
SLP UA INFO: Waiting for SLPRepl
SLP UA INFO: Waiting for SLPRepl
Service Type: SPEAKER
Service URL: service:speaker://s
```

Figure 27: The Room locator client application

4.6.6 Room locator server

The room locator server is a web application that accepts location requests from our mobile SIP user agent and executes a room locating algorithm to identify the most probable location of the user. The location request from the client contains an IR beacon number and the temperature and light intensity values formatted in XML as shown in Listing 2 above. The room database table contains mappings between IR beacon numbers and rooms and information about the last entrance to and exit from each room. These are used as inputs to the room locating algorithm. We have used real-time light and temperature readings from the different rooms to further improve the accuracy of the user's location determination.

The room locator server also provides a secondary functionality by monitoring and controlling the room registration using a simple web user interface. It enables us to easily register new rooms and their corresponding IR beacon mapping. The web user interface also let us monitor the activity of the locator service by presenting a history table - enabling the user to monitor the activity of the room locator server. We have found this feature very useful when developing and evaluating the system. Figure 28 shows the screen shot of the web user interface used to monitor the room locator server.



Welcome to Room Locator Service

New Room Registration

id:
name:
ir_beacon:
last_enterance:
last_exit:
Insert Cancel

Current Room Status

id	name	ir_beacon	last_enterance	last_exit
1	OpenArea	3	6/24/2009 15:12:15	6/12/2009 16:16:09
2	MINT	1	6/12/2009 16:18:12	6/11/2009 16:22:11
3	Desk	4	4/14/2009 17:38:37	4/14/2009 17:38:39

History

			Timestamp						Action by			
ID	Room ID	Beacon	Entrance	Exit	Device Light	Device Temp.	Room Light	Room Temp.	PIR	Light	Temp.	Final
477	1	3	6/24/2009 15:12:15	6/12/2009 16:16:09	403	3283	510	2533	entrance	unknown	unknown	entrance
478	3	4	4/14/2009 17:38:37	4/14/2009 17:38:39	294	2535	2471	2495	exit	unknown	unknown	exit
479	3	4	4/14/2009 17:38:37	4/14/2009 17:38:39	295	2539	2472	2496	exit	unknown	unknown	exit

Figure 28: The Room locator Server web interface

The locator server is implemented as an ASP.Net web application deployed on a Microsoft IIS 7.0 web server. Appendix G provides a partial listing of the C# implementation of this web application. The *Page_Load()* method will be executed when the web server receives an XML location request. The method extracts the beacon number and the light and temperature readings. The beacon number will be used to query the database server to identify the corresponding room number. The last entrance and exit fields will be used to estimate if the user just left or entered the identified room. The light and temperature readings will be compared (see section 4.6.3 for details) with that of the room in order to further improve the accuracy of location estimation. Finally the location (i.e. the room number) and action (i.e. exiting or entering) will be used to create a location reply. We also compute a certainty value (that ranges between 0 and 1) to describe how accurate this estimation is. The certainty value is computed based on the weighted information provided by the PIR sensor, IR beacons, and the light and temperature sensors. The decision made by the location server is also saved in the history table of the database server.

.

5. Trust Relationships & Media Redirection

By using the context aggregation framework we presented in the previous chapter, our SIP user agent is able to determine if it is entering or leaving a specific room. The next task will be to dynamically discover multimedia devices and services in a secure manner. We decided to use the Service Location Protocol (SLP) due to its advantages, as presented in section 2.2.3. The services provided by multimedia devices such as speakers, microphones, projectors, and cameras will each have a unique URI, which is a fully qualified SIP URI. Subsequently by using this SIP URI we can redirect media to/from the multimedia device.

It is important to note here that, before redirecting media to the discovered devices the mobile user should somehow establish some level of trust in the service to be provided by the discovered device(s). We propose a public key based authentication mechanism where the service advertisements will be digitally signed by the service providers. Redirection of media will only take place if the signature presented by the service provided is verified by a trusted third party. The trust relationship established in this way will also be used when exchanging the keying materials used to secure the actual media stream. In order to demonstrate the feasibility our proposal, we have implemented and evaluated a mobile SLP user agent that includes these functionalities. Before proceeding with the implementation details, we review the design choices we have made.

5.1 Service Discovery using SLP

In this section we will present two possible ways to use SLP and identify the most appropriate approach for our system. The basic SLP network client applications are modeled as User Agents (UA) and services are advertised by Service Agents (SA). As shown in Figure 29, the user agent issues a service request specifying the service the user is interested in. Service requests are sent to the SLP multicast address[17] 239.255.255.253. Service agents are implemented as a daemon that listens for service requests on the SLP port 427. Upon receiving such a request, SAs check to see if the service request matches the specification of the service that this device implements This matching includes comparing the service-type, scope, and predicate logic of the requested service to the service provided. When a match is found, the SA unicasts a service reply message. This service reply includes a URL for the service and other optional parameters.

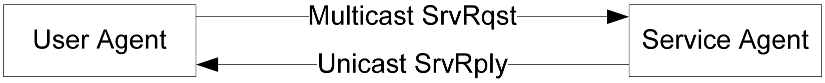


Figure 29: Basic SLP Network [8]

Alternatively in large networks, a Directory Agent (DA) may be deployed to provide caching functionality. SAs will register services they provide by unicasting service register

[17] In IPv6 network service requests are multicasted to a group IPV6 address between FF0X:0:0:0:0:0:1:1000 and FF0X:0:0:0:0:0:1:13FF. The value X is used to scope the request.

messages to the DA. In this scenario, UAs unicast their service request to the DA, which is responsible to perform the appropriate service matching operation. For this setup to function, UAs and SAs must identify their directory agent. This can be achieved by multicasting service request messages with a service type set to *directory-agent*. Alternatively the nearby directory agent can be learned using DHCP or by static configuration.

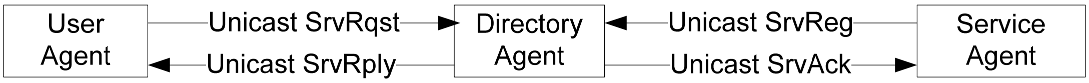


Figure 30: SLP network with a DA [8]

As application is intended to be used in a home or office network, due to the limited number of services that are likely to be deployed in such a network, it makes sense to deploy our application without a directory agent. However, in the future as the number of services available in the network increases, a directory agent could be deployed to improve performance.

5.1.1 Provisioning service discovery

Using the approach shown in Figure 29 all the service agents which are members of the SLP multicast group will receive the service request and each may send a reply to the requesting user agent. However, we are only interested in a reply from services in a particular room. One approach could be to use a user defined attribute (for instance location) to be a basis for service comparison. Therefore, the user agent can filter out services replies whose location attributes match the room we are interested in.

A better solution would be using SLP scopes to provide administrative grouping of services. By using this option, we remove the requirement to use a user defined attribute to group services and instead use the built-in mechanism of categorizing services. SLP allows a set of services to be assigned to a particular scope by a network administrator[18]. The client application will then be able to discover services within the scope they belong to. More formally this SLP scope is a SLP Scope Name Attribute. We propose that a set of services will be assigned to a particular scope based on their physical location. For instance camera and projector services in Grimeton will be assigned to the scope of Grimeton and speaker service installed in room6339 will be assigned to an SLP scope of room6339 and so forth (see Figure 31 below). Note that "room6339" is simply a symbolic string to refer to resources in room 6339. Note that the SLP scope is different from the scope of the multicast (for example as described in the previous footnote concerning IPv6 multicast scope).

When the mobile user agent makes a service request, the current location (identified using our context aggregation framework) will be used as a scope identifier. We believe this approach effectively delimits the service discovery without introducing a user defined attribute to be configured both on the service agent and on the mobile user agent.

[18] By default all SLP services belong to a *DEFULT* scope.

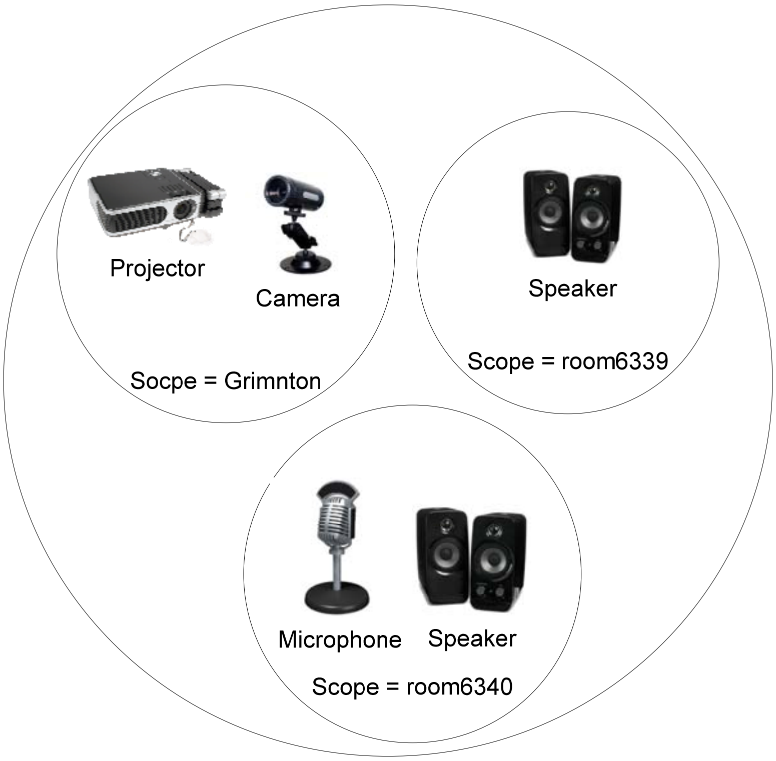


Figure 31: Grouping services using scope

5.2 Establishing a Trust Relationship

By introducing context-awareness to mobile multimedia communication, we can create a more enjoyable user experience allowing simple interaction with nearby devices and services. An interesting aspect of context-aware systems is their ability to provide personalized services to the user. This requires the user’s explicit consent in disclosing some of their contextual information (for instance their location and current activity). In order to obtain the user’s consent, users will have to *trust* the system. Trust in this context can take two forms. The first is that the users would like to be certain that inferences made about the user's situation are sufficiently accurate. For instance, assume that our context aggregation framework makes frequent incorrect determinations of the user’s location. This bothers the user with efforts to redirect media to devices which are not nearby. This will reduce the user’s satisfaction and eventually the user will not trust the system. Possible countermeasures could incorporate more accurate sensors and context sources. However, this in turn will need greater willingness of the user to disclose additional context information. Therefore a better solution would be to improve the system's accuracy by performing detailed context synthesis and learning based reasoning utilizing existing context sources. Eventually when the system’s accuracy has improved sufficiently, additional sensors and context sources can be introduced without losing the user’s trust in the system.

The second aspect of trust we are considering concerns privacy and security. In order to enhance the usability of our context aware SIP user agent we would like to assure the user that the devices they interact with are genuine. If we do not provide an appropriate mechanism for authentication, the following attacks can occur:

Unauthorized registration: The attacker registers non-existent or malicious services with the directory agent. This will result in the client application starting to interact with the malicious service or causing it to take it a long time to find a functional service.

Unauthorized deregistration: This is a more effective attack because it causes legitimate services to be deregistered, resulting in the client application not being able to locate services it is interested in.

Unauthorized DA: An unauthorized directory agent could take full control of the client service discovery. It would produce service replies with fake URLs, (pseudo-)randomly accept and drop service registrations, and so on.

Unauthorized SA: A malicious service agent could masquerade as a provider of a trusted service allowing it to gain access to the user's private information.

Consider the situation where an adversary sets up a fake speaker service agent to reply to service location requests by our SIP user agent. In the absence of an authentication mechanism for the user’s reply, the SIP user agent will start to redirect sound to the rogue device. This is considered a serious invasion of the user’s privacy which will significantly affect the usability of our system. It can also be a major security break unless the session key is changed before doing this redirection, since otherwise the rogue service could learn the key for this media stream and use this information to attack the session key.

To counter this problem we propose a public key based authentication mechanism. SLP version 2 allows an authentication block to be attached to SLP messages so that receivers can verify the authenticity and integrity of the message. SLP service agents digitally sign their service advertisement using their private key. SLP user agents and directory agents can verify the advertisement (specifically the URL and the corresponding attributes) using the public key of the service agent.

It is important to note here that protecting the integrity of the service URL does not provide a complete guarantee that the actual service provided is authentic. An authenticated service reply can only guarantee the integrity of the service URL (in our case the integrity of the fully qualified SIP URI). However, an adversary could easily setup an IP or DNS spoofing attack allowing any device to reply to this address. Therefore it is important to prove the authenticity of the device that provides the service at a different level (i.e., using user name and password, TLS, S/MIME, or IPSec). The next section will present how the trust relationship created by authenticating the SLP reply can be maintained all the way to protection of the actual media stream.

5.3 Secure Media Redirection

So far we have described how we can identify the user’s relative location (i.e. entering or exiting a particular room) and the multimedia devices available in the identified location. Based on the findings of Oscar Santillana's master’s thesis, we have decided to use the third party call control (3PCC) method to redirect media (see section 3.4). His thesis shows how to use 3PCC to redirect RTP media from a mobile node to local devices. We have extended this approach to provide media redirection functionality for secure streams (i.e. when using SRTP).

Our main contribution is providing a secure key transfer mechanism. In order to provide privacy and integrity protection, SRTP uses session keys (authentication and encryption keys and salt values). The session keys are securely derived from a single master key. Currently the SRTP RFC [21] defines a pseudo-random function based on AES-CM, which takes a 128 bit master key and produces a random value of up to 2^{23} bits used as a session key.

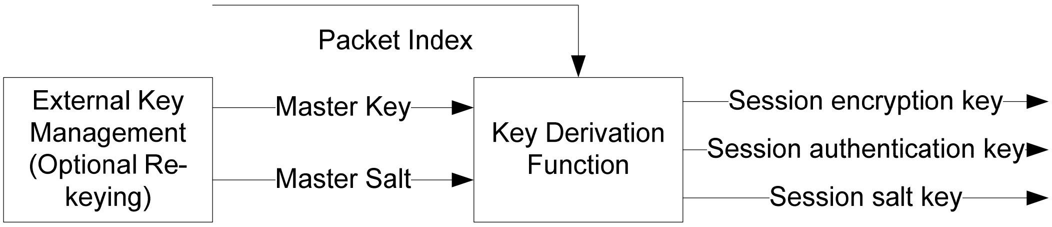


Figure 32: SRTP key derivation

Therefore the local device, to which we are redirecting the media stream, needs to access these session keys. For this purpose we have to securely transfer the master key used to derive the session keys as shown in Figure 32 (or we have to transfer a specific session key -- we will do this rather than provide the service with the master key, thus minimizing the damage that a rogue service could do - by limiting it to knowledge of the keys relevant to the media stream that it will have the RTP for).

In section 2.4.3 we presented key agreement protocols including MIKEY, the SDP crypto attribute, and the SDP key management extension. We have decided to use the SDP crypto attribute due to its simplicity. We propose to transfer the specific session key as part of the SDP of the final ACK message sent from the mobile node to the local node (see Figure 33). However, transferring the key materials in the clear makes no sense and we have to provide privacy and integrity protection of the key transfer. Here we can take advantage of the trust relationship we established during service discovery. The local device's SLP service agent signed the service reply using its private key during service discovery. The same public key used to verify the SLP service reply will now be used to encrypt the key transfer messages. This enables us to maintain the trust relationship we created during authentication of the service reply all the way to media protection. Furthermore, by using SRTP we protect against attackers who have successfully deployed an address spoofing attack on a trusted service provider. This means an adversary will not be able to decipher the SRTP media unless it posses the corresponding session keys.

Figure 33 shows the proposed mechanism for establishing trust with an example speaker service and performing a secure media transfer. After the mobile node issues an SLP service request, the speaker service agent will reply with the URL of its speaker service. The reply will be signed with the private key of the service agent. The mobile device can now verify the signature of the reply using the speaker's public key[19]. In order to establish an SRTP session with a speaker service, we need to transfer the master key to the speaker service encoded as a *crypto* attribute in the SDP of the final ACK message.

[19] Currently SLP does not provide any mechanism for distributing public keys. The public key to an SPI string mapping needs to be available both at the UA and at the SA.

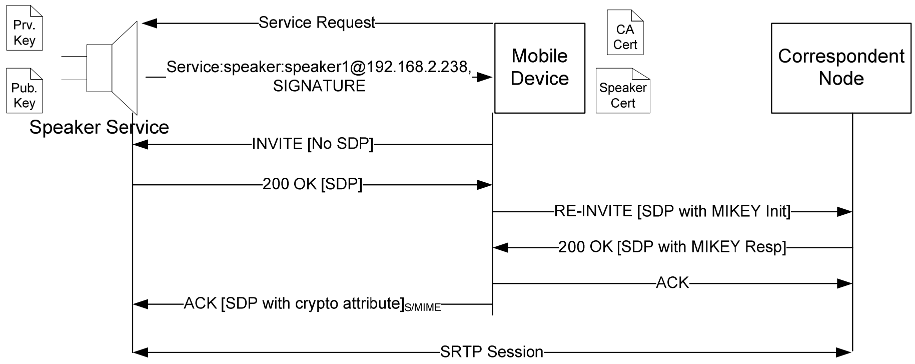

Prv. Key
Pub. Key
Speaker Service
Service Request
Service:speaker:speaker1@192.168.2.238, SIGNATURE
Mobile Device
CA Cert
Speaker Cert
Correspondent Node
INVITE [No SDP]
200 OK [SDP]
RE-INVITE [SDP with MIKEY Init]
200 OK [SDP with MIKEY Resp]
ACK
ACK [SDP with crypto attribute]S/MIME
SRTP Session

Figure 33: Trust establishment and secure media transfer

In order to provide end-to-end protection of the *crypto* attribute we used S/MIME. S/MIME is chosen for two reasons. First S/MIME provides end-to-end security for the contents of the SIP message. In comparison, TLS only provides a hop-by-hop protection of the whole SIP message. If we use TLS, then a trust chain must be established between the mobile node, the intermediate proxies, and the speaker service. In a context of an inter-domain environment, maintaining this web of trust is relatively difficult. One alternative could be using up-cross-down trust establishment to easy this burden [45]. S/MIME on the other hand provides encryption and integrity protection of part of the SIP header (excluding headers used for routing messages by intermediate proxies such as To, From, Call-ID, CSeq, and Contact) and the body of SIP message.

Secondly S/MIME certificates have the ability to assert an end-user identity. Unlike certificates used by servers, which assert the identity of the holder to a particular host name, S/MIME certificates assert that the holder is identified by the given end-user address. This address is the user name and domain part of the SIP URI (commonly known as address-of-record). Hence an S/MIME certificate signed by a trusted CA, provides us with proof that a given public key belongs to a particular address-of-record. This allows us to use the same certificate to protect the SDP of the SIP messages and assert the authenticity the URL of the service that we discovered using SLP (because our SLP URL contains the address-of-record of the service provider).

5.4 Implementation

5.4.1 Test bed and development environment

The test bed used to implement and test the service discovery, trust negotiation, and media transfer uses the machines and services presented in Table 7.

Table 7: Test bed for service discovery, trust negotiation, and secure media transfer

Machine	Service	Description
CCSIPAQ (Similar configuration to Table 3)	192.168.2.99:5060 - Minisip UA	Windows CE version of the Minisip user agent from [19] has been extended to implement 3PCC secure call transfer functionality. It is implemented using C++ and is integrated in to the existing Minisip code.
	192.168.2.99:ANY - SLP UA	An SLP user agent with the ability to authenticate service replies from service agents is implemented using C#.Net.
CCSLEFT (Similar configuration to Table 3)	192.168.2.238:5060 – SER server	SIP proxy and registrar server.
	192.168.2.238:35345 - Minisip UA	Linux version of Minisip user agent from [19] has been extended to implement 3PCC secure call transfer functionality. It is implemented using C++ and is integrated in to the existing Minisip code.
CCSMOTO(Similar configuration to CCSLEFT on Table 3)	192.168.2.4:427 – SLP SA	An OpenSLP speaker service agent is configured to reply for service requests from user agents. OpenSLP is also configured to digitally sign service replies to enable user agents build trust on discovered services.
CCSBEMNET(Similar configuration to Table 3)	192.168.2.90:5060 – JAIN SIP UA	A Java SIP user agent acting as a local device to which we are transferring the session to.

5.4.2 Configuring OpenSLP

OpenSLP is open source implementation of the version 2 of the SLP protocol [28]. It is written in the C programming language and is targeted for Linux systems. The current developmental version was 1.3.0. This version has been ported to win32 and various varieties of Linux platforms. For developers working in Java, version[20] 1.0 of the implementation has been developed in Java.

5.4.2.1 Installing OpenSLP

In our systems the OpenSLP daemon (*slpd*) has been used to act as a service agent. In order to provide authentication of service replies we have built OpenSLP with the following steps.

- Download the source tarball:

```
ccslabda:/home/bemnet# wget
http://prdownloads.sourceforge.net/openslp/openslp-1.3.0.tar.gz
```

- Untar the tarball:

```
ccslabda:/home/bemnet# tar -zxf openslp-1.3.0.tar.gz
```

- Extracted the directory and become root:

```
ccslabda:/home/bemnet# cd openslp-1.3.0
ccslabda:/home/bemnet# su
```

- Configure, make, and install the package. OpenSLP by default comes with security features disabled. In order to build with security feature we have to specify the "--enable-slpv2-security" flag to the configure command. **CAUTION**: Both the online user guide and the read me file in the tarball instructs the user to use the "--enable-security" flag which does not enable security. To add to the problem, the configure script does not check if we have provided a correct flag, it simply ignores flags that it does not recognize. To verify if the installation has security enabled, one should check as shown below. We have contacted the developers to correct this error.

```
ccslabda:/home/bemnet/openslp-1-3-0# ./configure –enable-slpv2-security
ccslabda:/home/bemnet/openslp-1-3-0# make
ccslabda:/home/bemnet/openslp-1-3-0# make install
```

- Check if OpenSLP is installed with security enabled.

```
ccslabda:/home/bemnet/openslp-1-3-0# slpd -v
      slpd version: 1.3.0
      compile options:
            debugging:        disabled
            predicates:             enabled
            slpv1 compatibility:    enabled
            slpv2 security:         enabled
```

5.4.2.2 Configuring OpenSLP with security support

OpenSLP uses three configuration files: - *slp.conf*, *slp.reg*, and *slp.spi*. The *slp.conf* file contains parameters that are used by our service agents. The set of parameters that we have configured for the speaker service agent are described below. All the other parameters are set to their default value.

- Scope will be use in our application as a specific logical location in the building. For instance SIP UAs located in room1 will specify that they are interested in services available in this room by specifying the corresponding room name as service scope. The

[2020] This version of OpenSLP does not include support for IPv6.

net.slp.useScopes attribute allow us to specify a comma separated list of scopes that the service should be assigned to. For example:

```
net.slp.useScopes = “grimeton, default”
```

- List of interfaces to be used by the SA. For example:

  ```
  net.slp.interfaces = “192.168.2.4”
  ```

- In order to include an authentication block in the service reply messages we should set the net.slp.securityEnabled parameter. This is done by specifying:

  ```
  net.slp.securityEnabled = true
  ```

The *slp.reg* file contains a list of services that are statically configured to be used by the SLP daemon (*slpd*). Since in our configuration we do not have a directory agent, each service agent will be configured in a similar way. The following example shows and slp.reg entry for the speaker service.

```
#Register the speaker service
service:speaker://speaker1@192.168.2.238,en,65535
scopes=room1
description= 7.1 Stereo Sound System
authors=bemnet
```

From this entry we can see that the service type is *speaker* and the URL for the service is *speaker://speaker1@192.168.2.238.* The URL specifies the SIP URI that our SIP UA will redirect the RTP media stream to. The scope attribute indicates the room this service is provided in. The string *en* indicates the locale the service uses (in this case English). The value 65535 indicates the lifetime of the service. In this case the service will be available as long as the *slpd* is alive. Currently OpenSLP does not have a mechanism to indicate persistent service registrations. When the *slpd*(running as a DA) dies, then all the dynamically registered services will be lost. There is a future plan by the OpenSLP maintainers to provide a persistent store of service registration information if the *slpd* dies unexpectedly.

When OpenSLP is compiled with security features, the *slp.spi* file is used by an SLP daemon to specify the keys used for signing and verifying SLP messages. This configuration file uses a Security Parameter Index (SPI) to create a mapping between a security context and the key files. An authenticator (in our case the UA), will specify the SPI when sending a request, so that the reply will be signed with the correct key. Below we can see the SPI entry we have used for our speaker service agent.

```
PRIVATE spi1 /etc/secure/privkey.pem
PUBLIC spi1 /etc/secure/pubkey.pem
```

One important shortcoming to be noted here is the protection of the private key. In most cases, in order to avoid unauthorized access, the private key will be encrypted with a pass phrase (for instance using triple DES). The current implementation of OpenSLP does not support encrypted private keys, which forces us to store the private key in the clear. Note that potentially this could be stored locally in an encrypted file system or in a Trusted Platform Module (see the work of the Trusted Computing Group).

SLP uses the Digital Signature Algorithm (DSA) to integrity protect SLP messages. In order to create the key pairs (shown above) we have used *openssl*. OpenSLP expects the key files to be supplied in PEM format. The following command will first create DSA parameters, and then create the key pair.

```
ccslabda:/home/bemnet# openssl dsaparam -out dsaparam.pem 2048
ccslabda:/home/bemnet# openssl gendsa -out privkey.pem dsaparam.pem
ccslabda:/home/bemnet# openssl dsa -in privkey.pem -pubout -out pubkey.pem
```

The above command will produce the private and public keys in *privkey.pem* and *pubkey.pem* respectively. It is important to note here that Windows does not support public keys in PEM format. In order to import the public key into the Windows key store, we have to convert the key to a compatible format (for instance PKCS#12). In the next section we will show how we performed this conversion and how we access this key from the Windows CE certificate manager using our SLP UA.

5.4.3 Implementing SLP UA for the iPAQ

This section presents the design decisions and the implementation of our SLP UA. The UA is implemented so that it will easily be integrated with our SIP user agent running on Windows Mobile 2003. The UA is developed as a reusable API so that others can use it in order to introduce dynamic service discovery into their application. A partial listing of the *SLPUserAgent* class, can be found in Appendix H. Currently the UA is able to handle two messages: – *service request* and *service reply*. The *findService()* method constructs the *service request* message by creating the SLP header with the function ID set to the value SrvRqst (i.e. equal to 1). The method sets the service parameters, supplied by the caller (including *service-type*, *scope-list*, and *predicate lists),* in the send buffer of the request message as specified in RFC 2608 [8]. Finally we set the SPI field of the buffer to indicate the security context. This value is used by the service agent to determine which private key to use when signing the *service reply*. Just before sending the buffer, a new thread will be spawned to receive *service reply*. The buffer is multicast to the SLP multicast group address on UDP port 427 (this port is reserved for SLPv2).

Upon receiving the service request, the SA will find the service entry that matches the parameters mentioned above. When the receiving thread gets a packet in reply, the *processPacket()* method checks if it is a *service reply* (by checking if the function ID is set to SrvRply, i.e. equal to 2) and performs the appropriate parsing. Most importantly we extract the URL[21], the SPI string, and the timestamp values. These fields (marked in Yellow in Figure 34) are used to compute the digital signature. After the service reply is authenticated (as described in section 5.4.3.1 on page 62), the *processPacket()* function will return a data structure containing the service URL. The code snippet in Listing 4 shows how to use the location reply we got from the room locator server in order to search for a speaker service using SLP.

[21] In the current version of the UA we only support a single URL per service reply.

```
LocationReply[] location = this.uploadRequest(roomLocatorURL, xml);
SLPUserAgent slpUA = new SLPUserAgent(location[0], "spi1", "en");
SLPServiceRequest slpRequest =
   new SLPServiceRequest("", ServiceType.SPEAKER, "");
SLPServiceReply[] slpReply = slpUA.findServices(slpRequest);
if (slpReply == null || slpReply.Length ==0){
   System.Console.WriteLine("No Service Found!");
}
else{
   System.Console.WriteLine("Service Type: " + slpReply[0].serviceType +
"\nService URL: " + slpReply[0].URL);
}
```

Listing 4: Searching for a speaker service using the SLPUserAgent class

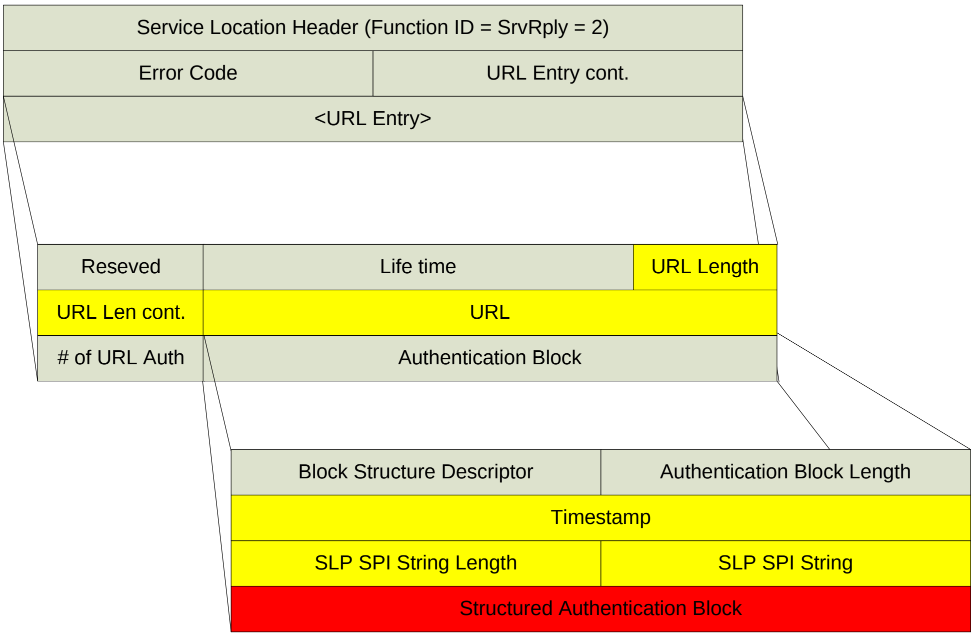


Figure 34: Service Reply Message (Yellow: Fields included in the signature, Red: the authentication block)

5.4.3.1 Authenticating Services

SLP version 2 introduced using authentication blocks with every URL entry to verify that the content transmitted is not modified and has been transmitted by a trusted party. The authentication has the data structure shown in Figure 34. The Block Structure Descriptor (BSD) identifies the structure of the authentication block. IANA maintains well known BSD values ranging from 0x0000 to 0x7FFF. The timestamp is an unsigned 32 bit value that represents the time when the signature expires. The start of the time epoch is 0h on 1 January 1970[22] .Service agents use this value to indicate the duration of the validity of the signature[23].

[22] This Timestamp will wrap back to 0 in the year 2106 and the time at which the time stamp is relative to resets.

The SPI field is a string that identifies the key length, algorithm parameter, and keying materials to be used to verify the digital signature. We use this value to retrieve the correct public key from a file.

For an implementation to support SLP v2 authentication, it must implement a digital signature algorithm with a SHA-1 authentication block (i.e. BSD = 0x0002). The Digital Signature Algorithm (DSA) is used to digitally sign SLP messages. The algorithm was proposed by the U.S. National Institute of Standards and Technology (NIST) and the signature calculation is specified in [46]. Compared to its counterpart RSA, DSA is only used for signing and it does not provide encryption. Additionally DSA is an ideal choice for signing SLP messages, because of the short size of the resulting signature it produces compared to RSA. This is important because when encapsulated in UDP, SLP messages can only be a maximum of 1400 bytes in size[24]. SHA-1 is used to compute the hash of the message to be signed (i.e. the field shown in yellow in Figure 34).

The default SLP signature format (i.e. BSD = 0x00002) conforms to X.509 v3 certificate format. It has the following three parts represented using ASN.1 encoding:

- The signature algorithm identifier (an OID):

  Id-dsa-with-sha1 (ASN.1 encoding of OID = 1.2.840.10040.4.3)

- The signature value:

  Binary ASN.1 encoding of r and s computed using DSA and SHA-1

- Certificate path

In order to extract the signature value from the ASN.1 encoded byte stream we have uses an open source ASN.1 parser from [47]. The code snippet shown in Listing 5 uses the *AsnParser* class to extract the *r* and *s* values (20 bytes each) of the X.509 signature values and saves them in a 40 byte array.

```
//Parse the ASN.1 signature to extract the r and s values.
AsnParser par = new slpua.AsnParser(signitureBytes);
par.NextSequence();
byte[] r = par.NextInteger();
byte[] s = par.NextInteger();
byte[] b40 = new byte[40];
Array.Copy(r, 0, b40, 0, 20);
Array.Copy(s, 0, b40, 20, 20);
```

Listing 5: Extracting the r and s values of the X.509 signature

In order to verify the signature we first have to compute the hash value of the fields whose signature is to be computed (including the SPI, URL, and timestamp fields). We have used the *SHA1CryptoServiceProvider* class that is present in the .Net security package to do this. When the *ComputeHash()* method is called a 20 byte hash value of these fields is

[23] OpenSLP sets the timestamp field to the maximum possible value i.e. 0xFFFFFFFF.

[24] Although not common, TCP can be used if larger SLP messages are desired. However, TCP traffic cannot be multicast - thus making this unsuitable for our use.

returned (see Listing 6). The *DSASignatureDeformatter* class implements the DSA signature verification algorithm. The *VerifySignature()* method takes as inputs the hash value and the signature to be verified. For the source code of the mobile SLP user agent we have implemented see Appendix H.

```
//Compute the hash value
SHA1 sha = new SHA1CryptoServiceProvider();
byte[] digest = sha.ComputeHash(authDataBytes);
//Verify the signiture
DSASignatureDeformatter verifier =
 new DSASignatureDeformatter(dsa);
verifier.SetHashAlgorithm("SHA1");
bool result = verifier.VerifySignature(digest, b40);
System.Console.WriteLine("****Verify****: " + result);
```

Listing 6: Compute the SHA1 hash value and verify the signature

5.4.4 Configuring SER

To provide SIP network element functionalities such as SIP proxy, registrar server, presence server, and redirect server we used open source software called Sip Express Router (SER) [20]. The various functionalities of SER are implemented as separately installable modules. These modules can be built and configured independently. The following steps are the tasks required to build and configure SER:

- Download the source code

```
ccsleft:/home/bemnet/ser# wget
ftp://siprouter.teigre.com/pub/ser/ser-2.0.0-rc1_src.tar.gz
```

- Unpack the tar archive

```
ccsleft:/home/bemnet/ser# tar xzf seri-2.0.0-rc1_src.tar.gz
```

- Compile SER with MySQL, presence and standard modules.

```
     ccslabda:/home/bemnet/ser# make group_include="standard
standard-dep mysql presence" all
```

- Install SER

```
     ccsleft:/home/bemnet/ser# make group_include="standard
  standard-dep mysql presence" install
```

- Download the *serctl* script used to manage the SER proxy from http://www.iptel.org/news/serctl and install to */usr/local/sbin/serctl*
- Create a new local domain (i.e., the IP address of the proxy server 192.168.2.238) using the *serctl*

```
     ccsleft:/usr/local/sbin/serctl #  ./ser_domain add test
192.168.2.238
```

- Add user (called ipaq) to the SER Registrar server

```
     ccsleft:/usr/local/sbin/serctl # ./ser_user add ipaq
     ccsleft:/usr/local/sbin/serctl # ./ser_uri add ipaq ipaq
@192.168.2.238
     ccsleft:/usr/local/sbin/serctl # ./ser_uri add ipaq ipaq
@192.168.2.238
```

```
ccsleft:/usr/local/sbin/serctl # ./ser_cred add ipaq ipaq test 192.168.2.238 ipaq
```

- Start SER using the configuration file in Appendix K

```
ccsleft:/usr/local/sbin # ./ser -f /home/bemnet/ser/auth-mysql-pa.cfg
```

5.4.5 Implementing secure media redirection in Minisip

In order to transfer an ongoing secure session to a local device we have extended the recommendations made by Oscar Santillana [4]. The Third Party Call Control (3PCC) functionality with a simple call flow shown in Figure 33 was implemented and integrated into to Minisip on both the HP iPAQ PDA (the mobile node) and on a Linux machine (the correspondent node).

On the HP iPAQ PDA we have built a Windows CE version of Minisip [19] with only a command line interface. The *MinisipTextUI* class provides a command line interface to make outgoing calls, answer incoming calls, and end active sessions. The communication between the user interface and the SIP layer is done using callback functions to send messages to a *message router*. When a message router receives a message it routes it to the appropriate destination (either a *gui, sip,* or *media*) using the *handleCommand()* method. In order to implement our session transfer functionality we defined a command called *localtrans* (see Listing 7). This will allow the user to issue this command from the command line (for example *localtrans speaker1@192.168.2.238*). When the message router receives this command from the *gui* it will call the *localTransfer()* method of the *sip* layer, which will initiate the 3PCC call flaw.

```
if ((command.size()>=10) && (command.substr(0,10) == "localtrans")){
    if (command.size()>=12){
        if (state!="INCALL" ){
            displayMessage("Not in a call!", red);
        }else{
            string uri = trim(command.substr(11));
            CommandString transCmd (callId,
            SipCommandString::local_transfer_requested,
            uri);
            //send command to message router
            callback->guicb_doLocalTransfer(transCmd);
            wprintf(L"Starting local transfer...\n");
        }
    }else{
        displayMessage("Usage: localtrans <userid>");
    }
    handled=true;
}
```

Listing 7: Code snippet from *MinisipTextUI* class for handling the *localtrans* command

The *localTransfer()* method starts by checking if the URI specified by the user is valid. Then the method creates a new SIP dialog (i.e. an instance of *SipDialogVoip* class) to represent the session between the mobile node and the local node. Next we create an invite command to be sent to the transaction user of the SIP dialog we just created (see section 2.3.2 to clarify the difference between dialog and transactions). The difference between the invite message to the local node and other nodes is that, the local node's invite message does not have an SDP body. Therefore we must tell the *SipDialogVoip* class that it should set an SDP when calling the *sendInvite()* method. This is done by saving the Call-Id of the dialog as shown in Listing 8. and checking if the Call-Id is different from *SipDialog::localCallID in* the *sendInvite()* method before setting the SDP.

```
string Sip::localTransfer(string &uri ){
    //...parse the uri
    //URI doesn't have the domain name part add it.
    if( !gotAtSign && id ){
        id->lock();
        uri += "@" + id->sipDomain;
        id->unlock();
    }
    SipDialogSecurityConfig securityConfig;
    MRef<SipDialogConfig*> callconf =
    MRef<SipDialogConfig*>(new SipDialogConfig(phoneconfig->inherited) );
    securityConfig = phoneconfig->securityConfig;
    MRef<Session *> mediaSession =
    mediaHandler->createSession( securityConfig );
    //create new SIP dialog with for the local device.
    MRef<SipDialog*> voipCall( new SipDialogVoip(sipstack,
                              callconf, phoneconfig, mediaSession));
    SipDialog::localDialog = voipCall;//save instance of the dialog
    SipDialog::localCallID = voipCall->getCallId(); //save call id
    //Add the dialog to the SIP stack
    sipstack->addDialog(voipCall);
    //Create invite command to be sent to the Transaction User (TU)
    CommandString inv(voipCall->getCallId(), SipCommandString::invite, uri);
    SipSMCommand cmd(SipSMCommand(inv, SipSMCommand::remote,
                                   SipSMCommand::TU));
    //Queue the message for the sip stack to handle it.
    sipstack->handleCommand(cmd);
    mediaSession->setCallId( voipCall->getCallId() );
    return voipCall->getCallId();
    //...
}
```

Listing 8: The *Sip::localTransfer()* method used to start the 3PCC call transfer

When the local node gets the invite message it will immediately reply with a 200 OK response. Upon receiving this response the SipDialogVoip::a3_callingnoauth_incall_2xx() method will be called. If the incoming call ID matches the call ID used for the local node (i.e. *SipDialog::localCallID*), then the mobile node saves the SDP of the response in a *SipDialogVoip::localSDP* static object. The mobile node uses this SDP when sending a re-invi te message to the correspondent node. Since the SDP from the local device does not have any key exchange attribute (i.e., the key-mgmt session level attribute), we defined the *SipDialogVoip::getReinviteSDP()* method to create the key-mgmt attribute and include it in the SDP from the local node. See Listing 9.

```
MRef<SdpPacket *> SipDialogVoip::getReinviteSDP()
{
    //get the sdp offer from the media session and extract the a=key-
mgmt:mikey line
    MRef<SdpPacket *> tempSdp = mediaSession->getSdpOffer();
    string keyMgmt = tempSdp->getKeyMgmt();
    //and add it to the one in localSDP
    SipDialogVoip::localSDP->setSessionLevelAttribute("key-mgmt",keyMgmt);
    return SipDialogVoip::localSDP;
}
```

Listing 9: The getReinviteSDP() method

When the mobile node gets a 200 OK from the correspondent node it will have to send an ACK to both the correspondent node and to the local node (confirming the first 200 OK). However, the ACK for the local node should include SDP from the correspondent node. More importantly it will need to include the keying material to be used to decipher the SRTP media. We have defined the *sendAckwithSDP()* method inside the *SipTransactionInviteClient* class to create the ACK message with the appropriate keying material. Appendix I provides a partial listing of this method. The method creates the crypto SDP media level attribute to securely transfer the master key used to derive session keys used to decipher the SRTP media. The crypto attribute has the following syntax.

```
a=crypto:<tag> <crypto-suite> <key-params> [<session-params>]
Where:      tag=0,1,2...
            crypto-suite= AES_CM_128_HMAC_SHA1_80
            key-param = "inline:" <key||salt> ["|" lifetime] ["|" MKI ":"
      length]
```

The *tag* field is a unique identifier of the crypto attribute. The *crypto-suite* specifies the encryption and authentication algorithm to be used. Currently there are three crypto suites (see Table 2 on page 14) standardized and registration of more new suite is managed by IANA. The *key-param* provides the keying material for the crypto suite in question. The string *inline* means that the key material is provided in the *key-param* itself following the (':'). The master key and the salt values are concatenated and encoded in *base64*. The *lifetime* is an optional field that indicates the maximum number of SRTP and SRTCP packets that can use this master key. By specifying Master Key Identifier (MKI), it is possible to identify which master key value to use by simple looking at the SRTP packet. As shown in Figure 4 on page 15, the SRTP packet contains an optional MKI field used to indicate the relevant master key[25]. The example below shows the SDP sent from the local device to the speaker service as part of the final ACK message.

```
v=0
o=- 3344 3344 IN IP4 192.168.2.238
s=Minisip Session
t=0 0
a=crypto:1 AES_CM_128_HMAC_SHA1_80
       inline:d0RmdmcmVCspeEc3QGZiNWpVLFJhQX1cfHAwJSoj|2^20|1:32
m=audio 30456 RTP/SAVP 0
c=IN IP4 192.168.2.238
a=fmtp:0 PCMU/8000
a=rtpmap:0 PCMU/8000
```

Finally before sending the ACK message to the local node, the SDP will have to be encrypted and signed using S/MIME. For encryption RFC 3853 [48] recommends using AES instead of 3DES. This is mainly because AES is considered to be faster and more memory efficient compared to 3DES, which makes it suitable for mobile and embedded devices [48]. For signing S/MIME uses RSA as its digital signature algorithm and SHA1 as the digest algorithm. The example shown in Listing 10 presents the full ACK message sent when calling the *sendAckWithSDP()* (see Appendix I). The content type of the ACK message is a multipart signed SDP. The first part (Binary blob 1) is an S/MIME encrypted SDP and the second part

[25] Note that SRTP has another method of identifying the master key used to process SRTP packet using the <"From", "To"> range of the 48 bit packet sequence number.

(Binary blob 2) is the signature of the SDP. The boundary between these parts is marked using an identifier specified in the *Content-Type* header of the ACK message. Note that the overall length of the content of the SIP ACK message is only 2158 bytes in size.

```
ACK sip:192.168.2.90:5060 SIP/2.0
Max-Forwards: 70
From: <sip:ipaq@192.168.2.238>;tag=20997
To: <sip:local@192.168.2.238>;tag=7709
Call-ID: 14206@192.168.2.99
CSeq: 901 ACK
Route: <sip:192.168.2.238;ftag=8664;lr=on>
Content-Length: 320
Content-Type: multipart/signed;boundary=75b3d73b4e24d3f6;\
              micalg=sha1;protocol=application/pkcs7-signature
Content-Length: 2158

--75b3d73b4e24d3f6
Content-Type: application/pkcs7-mime;
              smime-type=enveloped-data;name=smime.p7m
Content-Disposition: attachment;handling=required;filename=smime.p7
Content-Transfer-Encoding: binary

*****************
* BINARY BLOB 1 *
*****************
--75b3d73b4e24d3f6
Content-Type: application/pkcs7-signature;name=smime.p7s
Content-Disposition: attachment;handling=required;filename=smime.p7s
Content-Transfer-Encoding: binary

*****************
* BINARY BLOB 2 *
*****************
--75b3d73b4e24d3f6—
```

Listing 10: The final ACK message sent from the mobile node to the local node.

Up on receiving the final ACK message, the local device will use its private key to decrypt the SDP. The public key of the mobile node will be used to verify the signature. In the current implementation the public key files are preconfigured on both the mobile and the local node. This approach works well for limited deployment, such as in our test bed. However, in the future public keys with appropriate credentials could be distributed using a public key server.

6. Evaluation and Discussion

This chapter presents the two categories of measurements and evaluations that have been performed. A detailed discussion of the results and analysis will also be presented. The first set of evaluations performed mainly focused on quantifying the performance delays introduced by including context-aware features into the SIP user agent. For this purpose the room locator client was evaluated using the test bed described in section 4.6.1. Accuracy tests were made in order to understand how reliable the context framework is. These series of tests were performed to determine if the location determination made by the room locator server is sufficiently accurate for our target task (i.e., automatic configuration of the SIP user agent).

The second set of measurements focused on the performance of the trust establishment and media redirection features. The test bed presented in section 5.4.1 was used to evaluate how long it takes for the mobile device to authenticate service replies and to start redirecting the existing media stream(s).

6.1 Performance of the room locator client

The room locator is an application that enables a mobile SIP user agent to determine its current location (in this case room number) and action (either entering or exiting this specific room). In order to decide whether it is acceptable to include these features in the SIP user agent, we need to measure its performance; as if the performance is not adequate the user will not be happy with this service

Section 6.1.1 presents what aspect of the room locator client needed to be evaluated and section 6.1.2 discusses the findings and provides detailed analysis of the measurements that were made.

6.1.1 Measuring performance of the client

As presented in section 4.5, the room locator client listens for URLs broadcasted by IR beacons mounted either in the room (or near the entrance to the room) and initiates a room locator request. Before creating this request, we acquire the light level and temperature reading from the Wasa board installed on the iPAQ PDA. The XML request is sent to the room locator web server using HTTP. The response XML from the server describes the most probable location and action of the user (see section 4.6.5).

Figure 35 shows a sequence diagram representing the interaction between the room locator clients; along with the server and the corresponding delays.

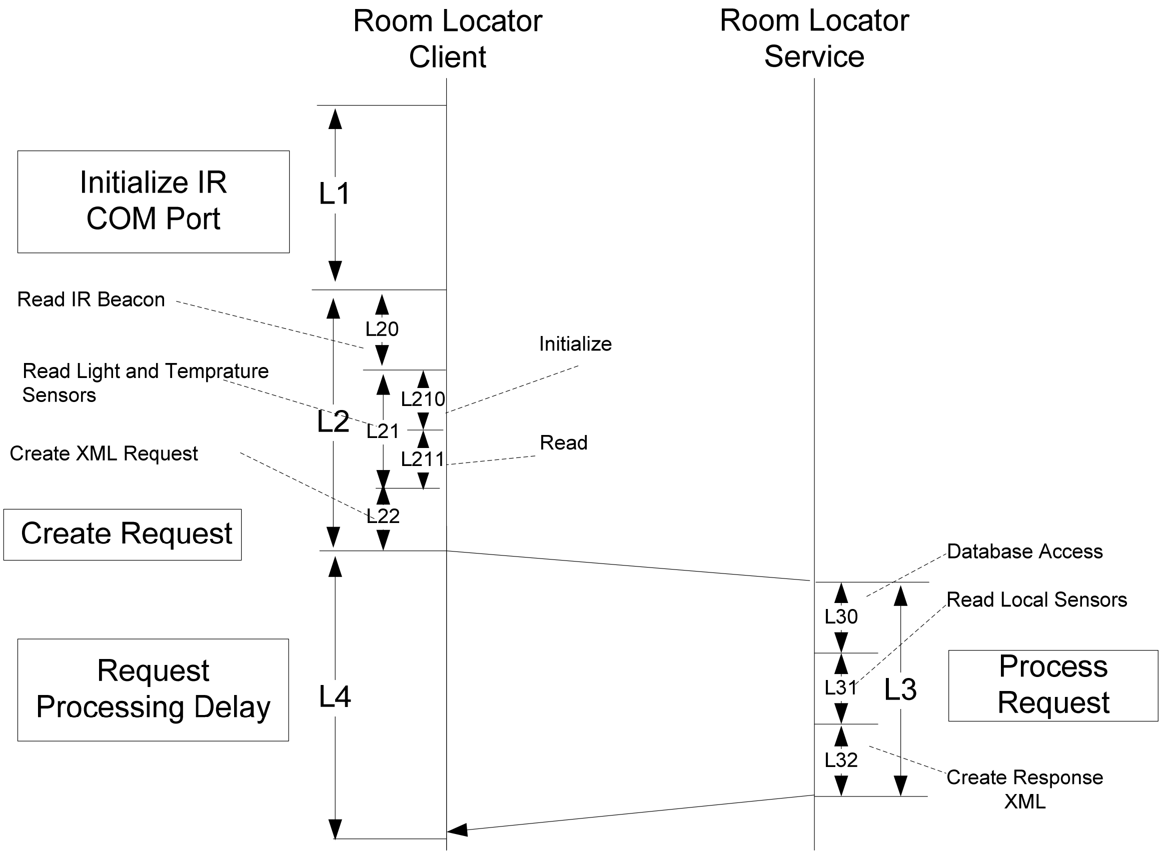


Figure 35: Delay breakdown for the room locator client

We have found the following six elements of the overall delay types interesting to measure and analyze. Table 8 shows these delays and describes what they account for. Note that in these measurements we assume that the PDA already has an established communication path to and from this server. The PDA dynamically discovers the IP address of the server using the SLP user agent described in section 5.4.3.

In order to determine these delays we have first used the date and time API provided in the .Net framework. The *DateTime* class provides methods to subtract two instances and return the result as a *TimeSpan* object we can to determine the duration in milliseconds as follows:

```
DateTime time1 = DateTime.Now;
//do something
DateTime time2 = DateTime.Now
TimeSpan span = time2.Subtract(time1);
double delay = span.TotalMilliseconds;
```

However, the result we obtained using this method on the HP iPAQ PDA made no sense (we were getting exceptionally high delay – in the magnitude of hours). After some investigation we learned that Windows CE devices exhibit strange behaviour when querying for current time with a resolution of milliseconds. A better time source is the system tick counter property – *Environment.TickCont*. The tick count is a 32bit signed value that represents the number of milliseconds elapsed since the last time the device restarted[26].

[26] This value rollovers when the device is kept running for 24.9 days; thus this is the maximum delay we can measure with only this method. However, this was not a limitation in any of our measurements.

Table 8: Description of room locator delays

Delay	Description
L1 - Initialize IR Port	The time it takes to open and set up the IrDA port (with bit rate = 9600bps). This is done only once when we start the room locator client.
L20 – Read IR beacon	The time it takes receive (58byte) the URL from the IR beacon.
L210 – Initialize	This is the time it takes to open and setup the serial port (with bit rate = 115200) to read light and temperature values.
L211 – Read	This is the time it takes to issue the AT command (twice) to read light and temperature values from the Wasa board. This includes the time it takes to decode the ASCII characters, parse the readings, validate, and optionally re-issue the AT command and receive new values (if either of the readings was invalid).
L22 – Create Request	This is the time used to load the XML file (only necessary for the first request after the room locator client begins), and to insert the values obtained from the IR beacon and light and temperature sensors.
L4 – Response Delay	This is the time elapsed between sending the XML location request and getting a response from the server. It includes the network delay and the processing performed on the server.
L30 – Database access	This is the time it takes to retrieve the mapping between IR beacon number and room number from the database server.
L31 – Access sensors	This the time elapsed to read the light and temperature readings of the Wasa board installed in the target room
L32 – Create XML	This is the time it takes the server to create the response XML containing the users location and corresponding action.

Tick counter approach of determining delay was used in the *IRBeaconReader* class (see Appendix C) to determine the six delays listed in Table 8. We performed five different test runs with six location requests per test. A test run in this context refers to starting the application; and making a request refers to the client application making a XML location request to the server when walking under one of the IR beacons mounted on the ceiling as shown in Figure 22 on page 38. Table 9 summarizes the delays for the 5 test runs and Figure 36 plots the delay for each test run.

Table 9: Room locator client delay measurements (5 test runs & 6 requests per test run). The last two rows correspond to the mean and standard deviation *excluding* the first request in each run (as this first timing includes all of the initialization times which do not re-occur in later requests in a given run).

	L1(ms)	L20(ms)	L210(ms)	L211(ms)	L22(ms)	L4(ms)
Run 1	165	65	144	122	496	1716
		54	105	58	8	486
		53	109	81	8	579
		54	105	80	8	587
		53	103	80	9	625
		56	106	82	9	574
Run 2	158	69	146	150	495	2359
		56	104	85	8	1429
		55	101	82	7	964
		56	108	62	9	852
		60	103	84	7	470
		55	104	110	8	937
Run 3	168	70	131	117	496	1796
		51	104	80	7	474
		56	101	85	7	637
		56	104	87	8	580
		55	100	78	8	807
		56	107	80	8	975
Run 4	160	69	144	116	496	1869
		55	106	85	7	2898
		60	113	84	8	664
		57	105	81	8	935
		57	105	86	9	548
		59	110	81	9	1372
Run 5	163	70	143	111	499	4412
		55	104	81	8	744
		56	102	85	8	695
		54	100	81	8	981
		55	102	82	8	898
		57	105	73	10	548
Mean	**163**	**58**	**111**	**88**	**89**	**1114**
Standard Dev.	**4**	**5**	**14**	**19**	**185**	**861**
Mean-Intermediate		**56**	**105**	**81**	**8**	**850**
Std Dev-Intermediate		**2**	**3**	**9**	**1**	**497**

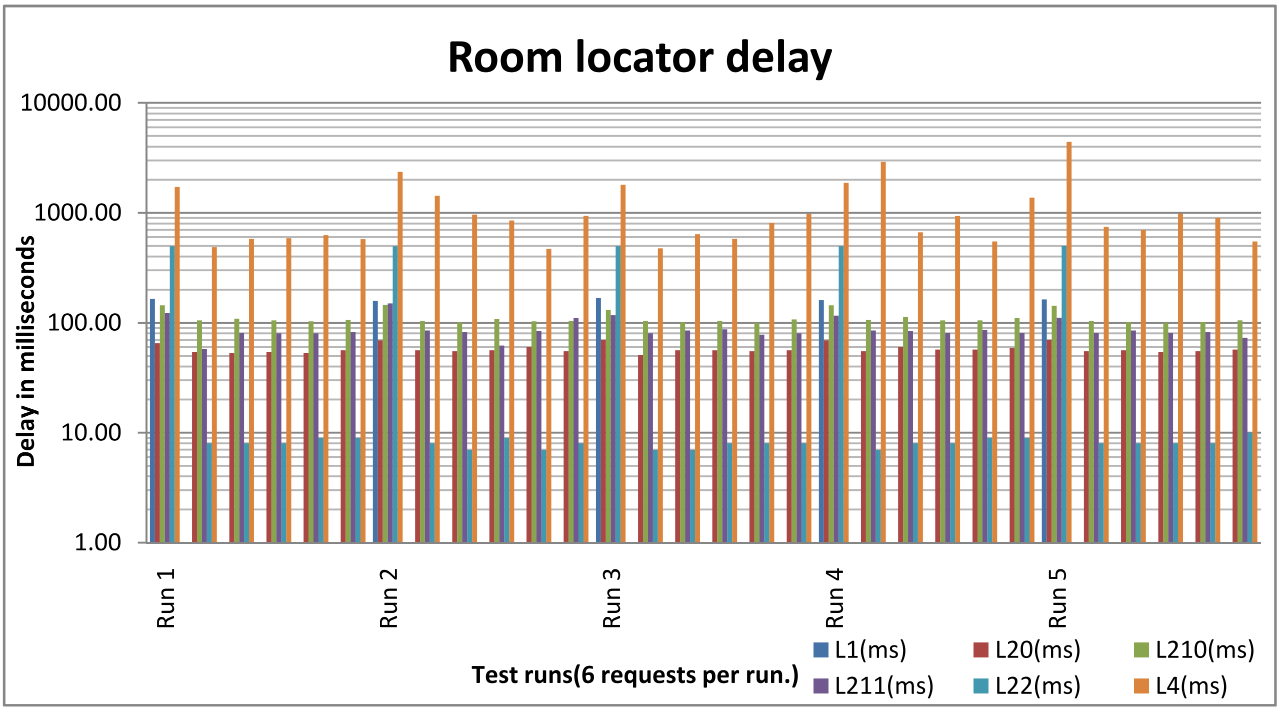


Figure 36: Comparison of the room locater client delay.

6.1.2 Performance analysis for the locator client

The delay specified as L1 is the time it takes to initialize the IrDA port. From Table 9 we can observe that the delay ranges between 158 ms to 168 ms with a mean equal to 163 ms and standard deviation of 4 ms. This delay is mainly due to the time required to open the IrDA port, which is accessed using an emulator serial port (COM2 on the iPAQ PDA). It also includes the time it takes to set the baud rate (which is equal to 9600bps) and other parameters as described in section 4.6.2. We believe that this delay is acceptable, because the IR port initialization only takes place once when we start the room locator client. As shown above any subsequent location request in a given test run does not experience this delay.

The delay specified as L20 is the time it takes to read the URL broadcast by the IR beacon. Each of the URLs had a total size of 58 bytes. As we have mentioned in section 4.6.2 on page 40, it takes two read operation to read the full URL using our configuration of the serial port API. The first read operation returns the first 32 bytes, and the next read operation returns the remaining 26 bytes. We experienced an average delay of 56 ms (as shown in the Table 9). Ideally at 9600bps, we expect to read the whole URL within 48.33 ms. The additional delay of 7.31 ms includes time for receiving framing bits. Because we are using 8-N-1 connection[27], for every eight bits transmitted, two additional framing bits are also transmitted - thus the total time for the 56 bytes with framing would be expected to be ~60ms. Note that since we might not be reading at the start of the beacon being transmitted, we expect to wait on the average one half of the beaconing period to see a beacon and to wait at most one beacon period to see a whole beacon (note that the beacon continuously transmits its URL - so the beacon time is simply the time it takes to transmit the URL). The transmission time

[27] 8-N-1 stands for (8) data bits, no (N) parity bit, and one (1) stop bit.

for the URL being sent twice would be roughly 120ms. We are not sure where the additional ~43 ms comes from, however some of it is probably due to the events driven processing used to receive the characters via the serial port driver.

The delay specified as L210 is the time it takes to initialized the serial port used to read the light and temperature readings from the ADC. The Wasa board is connected to the iPAQ PDA using a USB interface. In order to obtain the readings we open the virtual port once. We have determined that the time it takes to open the virtual serial port is around 110.80 ms. This delay includes the time it takes to configure the serial port parameters. It was surprising to learn that the delay to initialize the serial port is larger than the time it takes to read the actual light and temperature values (i.e. delay L211). Again, we believe that this delay is acceptable considering the fact that we only initialize the port once and reuse it for both readings.

The delay specified as L211 is the time it takes to read the light and temperature values from the Wasa board. The virtual serial port used to connect the Wasa board with the PDA was set to bit rate of 115200bps. The total time it takes to issue the AT command to access the ADC in order to read both light and temperature values was determined to be 88.30 ms. In comparison to Thor Hådén's report [49] in sampling data from Wasa board, our delay is significantly longer. His findings indicate that it was possible to collect 344 samples per second (i.e., each reading took only 2.9ms). The extended delay in our case is mainly because our delay not only includes the time it takes to read the sample using the AT command, but it also includes the time it takes to write to the serial port, read the response, parse the ADC reading, validate the reading, and possibly re-issue the AT command (if the value read is invalid). We observed that frequently the AT commands return invalid ADC reading (i.e. less than 0 and greater 4095 – because we have a 12 bit ADC we know that such values cannot be valid). We also noticed that if we issue the AT commands without a gap in time between them, the room locator client application crashes and quits[28]. Therefore we included a 100ms delay between AT commands. However this delay is subtracted from the data presented in Table 9, as this delay is a programmed delay and not a delay in the operation we are measuring. In order to further reduce the delay of creating location requests, the room locator client could be improved by reading the URL from the IR beacon and accessing the light and temperature values in parallel in separate threads.

The delay L22 is the time it takes to create the XML location request. Table 9 shows this delay is very high (above 490 ms) for the first requests within a test run. This is because for the first request the XML, the request template is loaded from a file into memory. All the subsequent requests have a lower delay (about 8 ms) because we do not access the local file system. The delay specified as L4 is the round trip time of sending the XML request to the room locator server and getting a XML response. We have determined that on average it takes about 850 ms to process location requests. This delay includes the round trip time uploading the request, accessing the room database, reading the room's light and temperature sensors, and downloading the response.

In the current version of the room locator server, we can only handle a single location request at a time. This is mainly because every location request accesses the serial port on

[28] The specific cause of this crash and alternative solution could be investigated in the future.

which the Wasa board is installed. This means that while the room locator web server is accessing the light and temperatures sensors, subsequent request will be blocked. This will be a bottleneck when the number of requests increases. A better approach could be implementing a separate process which continuously acquires new light and temperature readings and makes the latest values available to the room locator server. Thus essentially all of this delay could be removed from the overall L4 delay. Additionally, it should be noted that in a real installation there would probably be separate sensing of the room light level and temperature for each room - rather than the single sensor system used in this prototype.

In order to help us further improve the location determination delay we performed additional tests. On the room locator side, we have measured the time it takes to establish the database connection, access the light and temperature readings, and prepare and send the response. We can observe that establishing the database connection takes 15 ms for the first request. However, subsequent requests utilize this existing connection. The delay T31 is the time it takes to access the Wasa board. As we described earlier this delay can be parallelized with the location determination, because accessing the physical sensors can be implemented in a separate process. For example a separate process which accesses the light and temperature readings could transmit the latest values to the room locator service. This will considerably reduce the location determination delay and making our approach very scalable.

Table 10: Server side request processing delay

	T30(ms)	T31(ms)	T32(ms)
	15	218	110
	0	218	110
	0	218	109
	0	218	94
	0	203	125
	0	218	125
	0	203	124
	0	219	109
	0	203	109
	0	219	125
	0	219	125
Std. Dev	**5**	**7**	**10**
Average	**1**	**214**	**115**

Figure 37 shows how the delay of accessing light and temperature sensors with the request processing task. On the client side the tasks L210 and L211 can be parallelized with the request processing task. This will reduce the delay by 186 ms. On the server side, about 214 ms of the delay can be saved. Therefore if we introduce a parallelize approach a total of 400 ms can be removed, leading to a total L4 delay of roughly 450 ms -- making the location service more scalable. Figure 37 shows the overall time line with parallelization in both the client and the server.

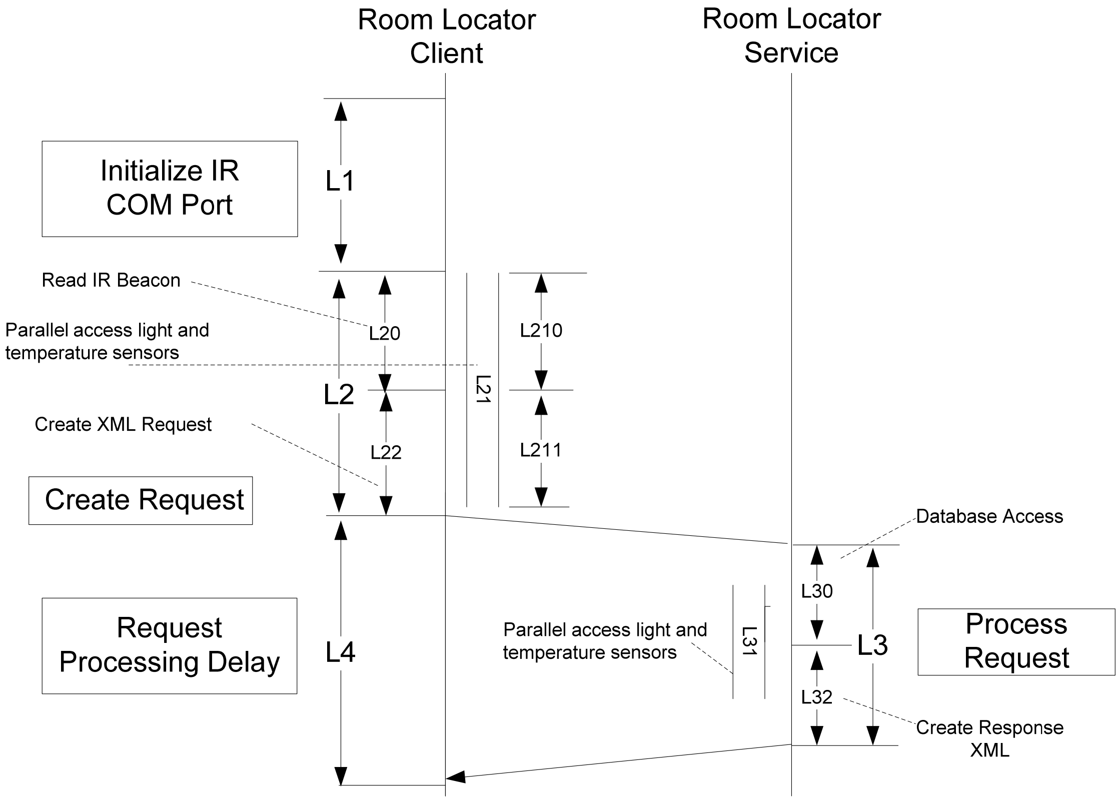


Figure 37: Parallelizing the light and temperature sensors with request processing.

6.2 Accuracy of the room locator server

In order to determine if the decisions made by the room locator server have an acceptable level of accuracy, we conducted accuracy measurements. The objective of these measurements was to determine how accurate the context aggregation framework is (see section 4.5). We also wished to understand how the accuracy would be affected when incorporating additional sources of context information such as light and temperature sensors. For this purpose we created a number of scenarios (presented in Table 11), where the user walks by our target rooms (room 6339 and room 6340 – see Figure 21 on page 37) and we collect the location decision on the server side using the history database.

Table 11: Scenarios for determining the accuracy of the room locator server

Scenario	Available Sensors	Expected Action
Entering room 6339	PIR, IR beacon, light and temperature sensor	Redirect media to speaker service (sip:speaker1@192.168.2.238)
Exiting room 6339	PIR, IR, beacon, light and temperature sensors	Redirect media to mobile node
Entering room 6340	PIR and IR beacon	Redirect media to speaker service (sip:speaker2@192.168.2.238)
Exiting room 6340	PIR and IR beacon	Redirect media to mobile node.

As it can be seen in Table 11, room 6339 is equipped with light and temperature sensors in addition to the IR beacon and room occupancy sensor. Generally we expect to have a higher certainty when we have input from the additional sensors. To test this hypothesis we have used a weighted certainty measurement. The certainty value is a number that ranges between 0 and 1. A certainty value of 0 indicates that the location of the user is unknown and certainty value close to 1 indicates that there is a near certainty that the user is located in a given room and is performing the indicated action (entering or exiting). Because the different sensors provide different levels of accuracy in determining the location and action of the user, the certainty value is calculated by adding the weighted probability values from different sensors. Location determined using the IR beacon has a higher weight (in this case we have arbitrarily set this to 0.40) because it is explicitly placed by the administrator at each specific room and the same administrator enters the mapping between the IR beacon numbers and the room number into the database of the room locator server.

Therefore if the location of the user is determined only using the IR beacon, the certainty will be 0.40. If the room status information published by the PIR sensor matches the previous decision, then the certainty value will be increased by 0.20. The light and temperature sensors each have a lower weight of 0.15. The comparison between readings from the device and the room is done based on the maximum difference between the ADC readings from two WASA boards exposed to the same light and temperature sources. The maximum difference between light sensors ADC reading is determined to be 30 out of 2^{12} values and for that of the temperature sensor we determined it to be 20. Note that here were are using the uncelebrated readings from the ADCs and not a calibrated value in degrees Kelvin for temperature or lux for the light level.

Table 12 shows the data collected from the room locator server for the above four scenarios. For each of the four scenarios we performed five consecutive tests. For the first two cases, three out of the five decisions were correct (i.e. 60% accuracy) with equal level of certainty (i.e. 0.60). Since we do not have light and temperature sensors in this room, the accuracy of the decision for these tests depends completely on the status update from the PIR sensor and the IR beacon. Xueliang Ren reported that he was able to obtain 60% accuracy when setting the PIR threshold to around 0.80 (the same value we have used) [32]. For scenarios three and four, we have been able to obtain a higher accuracy because we have used the light and temperature sensors mounted in room 6340. The certainty value of 0.75 indicates that the action decision made (i.e. exiting or entering) is supported by the PIR (with a certainty weight of 0.60) and by one of the sensors (additional certainty weights of 0.15). When both sensors support the PIR's decision (as in test 13) a much higher certainty of 0.90 is achieved. The agreement between our certainly measures and the actual situation was much better when using the data from these additional sensors.

During our test we detected a slight difference between the light sensor readings when exposed to the same light source. This resulted in decrease in the accuracy of the location estimate. To further investigate this problem we exposed the Wasa board to incandescent and fluorescent light source simultaneously. The results showed in Figure 38 shows that there is a fixed variance in the ADC readings.

Table 12: Decision made by the room locator server for the four scenarios (5 tests per scenario). E=Entrance, X=Exit, and U=Unknow

					Device Sensor		Room Sensor		Action				
ID	**Room**	**IR Beacon**	**Last Entrance**	**Last Exit**	**Light**	**Temp**	**Light**	**Temp**	**PIR**	**Light**	**Temp**	**Final**	**Certainty**
1	6339	4	6/11/2009 11:38	6/11/2009 11:23	341	2543	N/A	N/A	E	U	U	E	0.60
2	6339	4	6/11/2009 11:58	6/11/2009 11:39	722	2575	N/A	N/A	E	U	U	E	0.60
3	6339	4	6/11/2009 13:18	6/11/2009 12:01	362	2575	N/A	N/A	E	U	U	E	0.60
4	6339	4	6/11/2009 13:18	6/11/2009 14:38	309	2531	N/A	N/A	X	U	U	X	0.60
5	6339	4	6/11/2009 13:18	4/14/2009 14:38	323	2599	N/A	N/A	X	U	U	X	0.60
6	6339	4	6/11/2009 13:23	6/11/2009 14:52	315	2619	N/A	N/A	X	U	U	X	0.60
7	6339	4	6/11/2009 14:56	6/11/2009 14:52	2627	2615	N/A	N/A	E	U	U	E	0.60
8	6339	4	6/11/2009 14:56	6/11/2009 15:01	319	2615	N/A	N/A	X	U	U	X	0.60
9	6339	4	6/11/2009 15:05	6/11/2009 15:01	311	2607	N/A	N/A	E	U	U	E	0.60
10	6339	4	6/11/2009 15:05	4/14/2009 17:12	2607	2607	N/A	N/A	X	U	U	X	0.60
11	6340	3	4/12/2009 17:38	4/12/2009 12:38	356	2611	363	2558	E	E	U	E	0.75
12	6340	3	4/12/2009 17:54	4/12/2009 17:39	2633	2607	361	2560	E	U	U	E	0.60
13	6340	3	4/12/2009 18:09	4/12/2009 18:09	358	2569	361	2564	E	E	E	E	0.90
14	6340	3	4/12/2009 18:09	4/12/2009 18:25	360	2572	367	2568	X	E	E	X	0.30
15	6340	3	4/12/2009 18:41	4/12/2009 18:25	2321	2571	593	2567	E	U	E	E	0.75
16	6340	3	4/12/2009 18:38	4/13/2009 11:38	355	2609	364	2553	X	X	U	X	0.75
17	6340	3	4/13/2009 11:39	4/13/2009 11:56	352	2575	368	2528	X	X	U	X	0.75
18	6340	3	4/13/2009 11:58	4/13/2009 12:14	2655	2567	2567	2558	X	U	U	X	0.60
19	6340	3	4/13/2009 11:58	4/13/2009 12:32	389	2579	359	2557	X	X	U	X	0.60
20	6340	3	4/13/2009 11:58	4/13/2009 12:50	396	398	367	2558	X	X	U	X	0.60

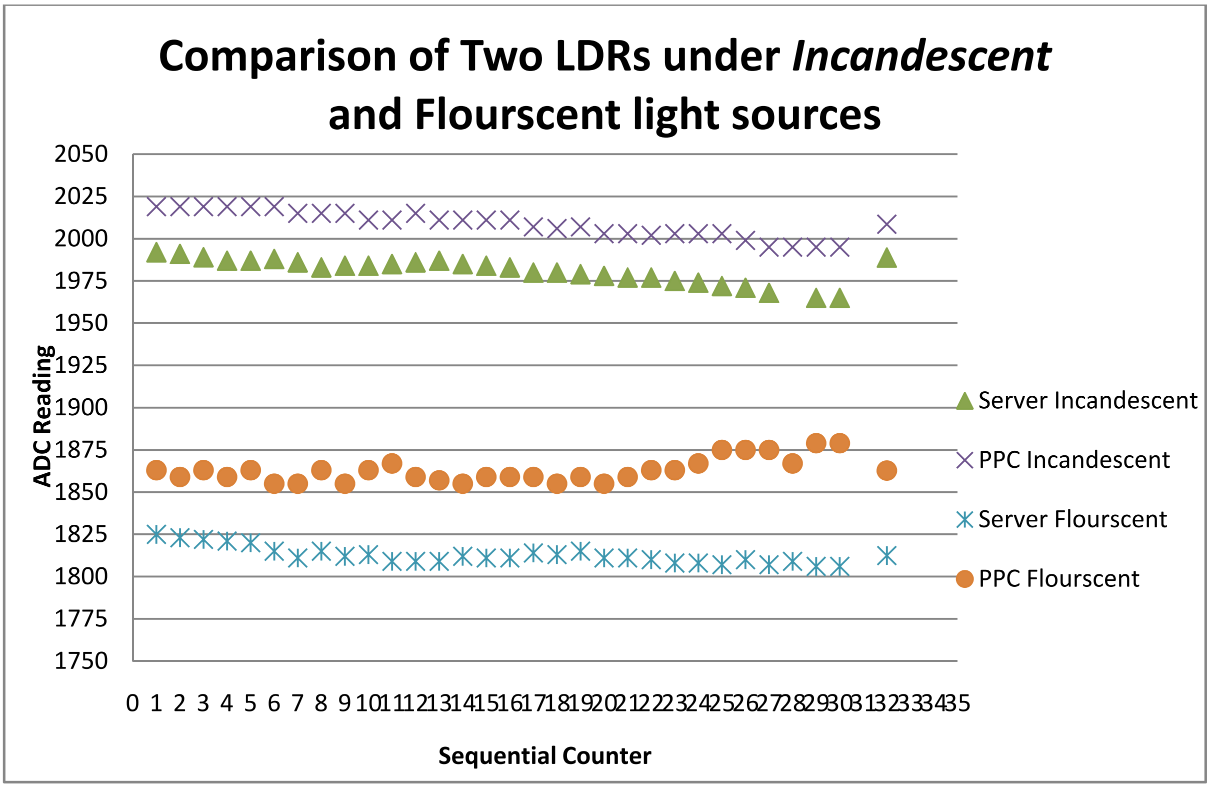


Figure 38: Comparison of LDR readings from two Wasa boards

On average the variance can be up to 50.3 of the ADC reading for a fluorescent lamp and 19.70 for incandescent bulbs. These offset values were used when using the light sensor reading is compared by the location server. Note that initially we did not calibrate the light sensors or temperature sensors.

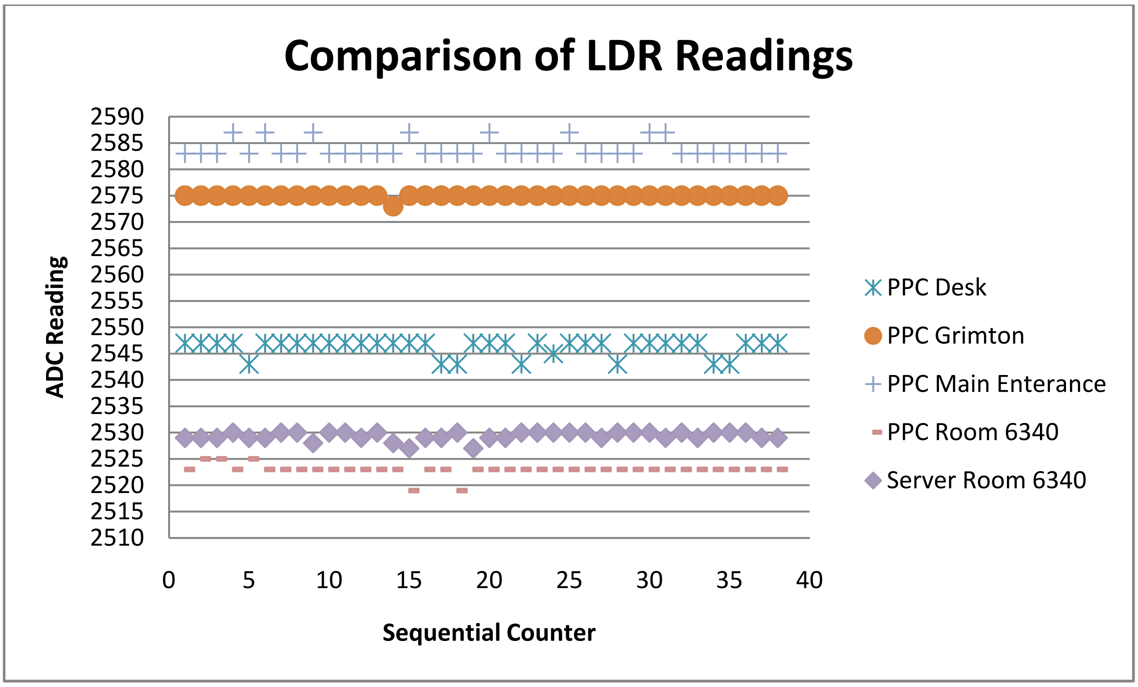


Figure 39: Comparing the LDR reading for various reference locations.

Figure 39 shows the comparison of LDR readings at different reference points in the lab. The readings marked PPC are collected using the HP iPAQ PDA; while the one marked Server was collected using the Wasa board mounted in room 6340(see Figure 21 on page 37 for floor plan). From the plot above we can clearly see that the reading collected in room 6340 is very close to the one we obtained when the user is in the same room.

The results we obtained clearly show that the accuracy framework increase when additional sensors such as IR beacons, light intensity and temperature sensors are incorporated. In comparison to previous thesis projects that use only the PIR sensor to determine room occupancy information [3; 32], we have been able to obtain 20%-30% increase in accuracy of location determination. Therefore the aggregated information gathered from the additional sensors is able to provide better discriminating effect – hence improving the accuracy of location determination. In the future, to further improve the accuracy of the context aggregation framework, additional sensors and contextual sources such as RFID tag readers could be incorporated.

6.3 Secure media redirection delay

In order to evaluate the performance of the secure media redirection technique, we conducted five test runs of the setup presented in section 5.4.1. The measurements to be performed will allow us understand the delay in performing the secured third party controlled call transfer. The SIP messages during the initial call establishment with the correspondent and the media redirection to the local node are captured at the SIP proxy using Wireshark [50]. In order to analyze the delay between the messages we have used the SIP Scenario Generator tool [51]. The call flow in Figure 40 was generated using this tool. The five unique colors in the call flow correspond to individual SIP dialogs. The first three dialogs are registration of the three user agents (local, correspondent, and mobile nodes). The initial session establishment with the correspondent node is indicated with a green color and the media redirection to the local node is indicated with a lime color. We can observe that the re-invite transaction with the correspondent node is performed within the same dialog as the initial call setup and hence the same color is used.

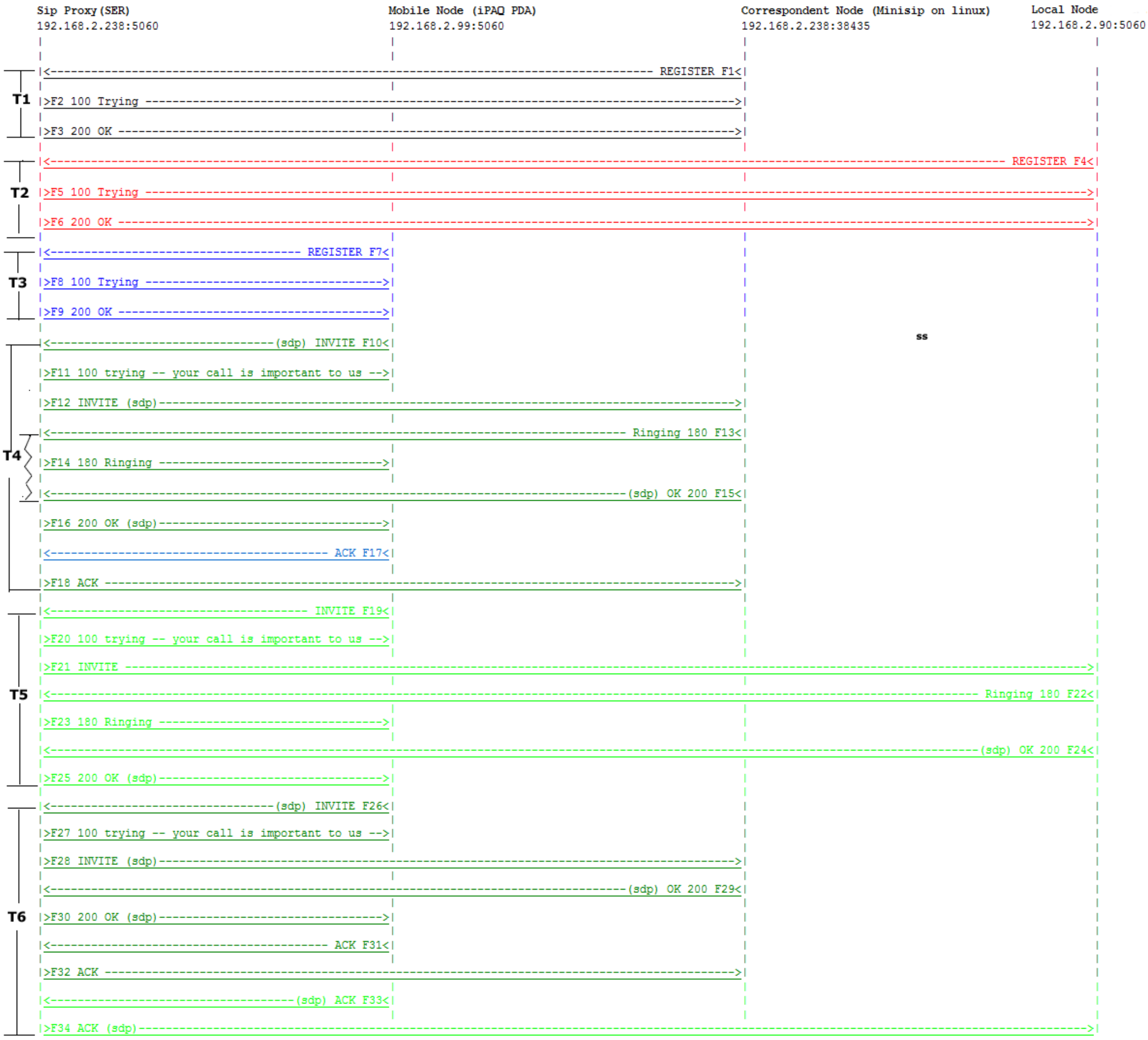


Figure 40: Call flow for redirecting media to a speaker service

For this purpose we have defined and measured six different delays. The first three delays (T1, T2, and T3) are latencies of registering the correspondent node, the local node, and the mobile node respectively. The delay specified as T4 is the time it takes to establish the initial session between the mobile node and the correspondent node. The ringing delay on the correspondent side (i.e. the time before the user accepts the call) has been eliminated by subtracting the delay between 180 Ringing and 200 OK as shown in Figure 40. The delay specified as T5 is the time it takes to establish a new session with the speaker service. For this delay we did not have to remove the ringing delay because the local node answers (i.e., replies to the invite message) almost immediately. The last delay T6 represents the re-invitation session with the correspondent node. The correspondent node's user agent (i.e. Minisip) code has been implemented such a way that incoming RE-INVITE messages are replied immediately to redirect the media. These times are summarized in Table 13

Table 13: Delay measurements for media redirection

Test	**T1 (ms)**	**T2 (ms)**	**T3 (ms)**	**T4 (ms)**				**T5 (ms)**	**T6 (ms)**		
				T4 Total	*Ringing*	*T4 Retrans*	*T4 Net*		*T6 Total*	*T6 Retrans*	*T6 Net*
1	2	2	1	4898	2956	1506	436	98	1209	500	709
2	2	2	1	4163	2979	650	534	98	1372	503	869
3	3	1	2	4646	2584	1506	556	96	1300	501	799
4	2	2	1	3528	2482	502	544	98	1219	502	717
5	2	2	3	3716	2685	496	535	97	1344	503	841
Mean	**2**	**2**	**2**				**512**	**97**			**787**
Standard Dev.	**0.5**	**0.5**	**1**				**48**	**1**			**72**

On average the registration delays range between 1.6 ms - 2.2 ms. Note that this delay only corresponds to the latency of processing the registration on the SER registrar server. Note that in our setup the registration messages are not authenticated. In order to avoid call hijacking attacks, in an actual deployment each registration should be authenticated. Determining the delay T4 (i.e. establishing a session with the correspondent node) requires more careful investigation. Simply subtracting the delay between message F10 and F18 in Figure 40 gives us a very high delay in order of 3-4 seconds -- marked as T4 total in Table 13. Therefore in order to determine the actual time required to establish the initial session, we have to eliminate the ringing delay and the possible retransmission of the 200 OK. From the raw captured data we have observed the correspondent node (i.e. Minisip) retransmits the 200 OK frequently (2-3 times) until it gets ACK from the mobile node. After eliminating these two delays the time it takes establish the initial session was determined to be 521ms.

Johan Bilien *et.al* performed measurements to evaluate the delay of establishing a secure session using Minisip [52]. Their results show that it takes about 30ms to process SIP messages and additional 30ms to verify the MIKEY messages and perform a policy check[29]. Note that his delays only refer to the delays on the user agents and do not include the network latency. However, as all of the nodes were connected by a local high speed LAN and WLAN infrastructure we do not believe that the network latency is significant in any of these measurements. We have noticed that a major portion of the delay (about 358ms) occurs on the mobile node when receiving a 200 OK from the correspondent node and before sending the ACK message. At this point it is not really clear why it takes this long to acknowledge 200 OK responses. The delay to invite the local node to the session is relatively very short. On average it takes about 97 ms. This is because the local node immediately replies with a 200 OK. Compared to T4, this delay does not have the MIKEY message processing latency since at this point no key exchange has been performed with the local node. On the other hand the delay T6 has considerably higher delay. The average delay we measured is 787 ms. Compared to the initial session establishment delay (i.e., T4), the added latency accounts for the additional ACK message sent to the local node (confirming the 200 OK in T5).

[29] Their test bed was based on two Linux boxes (rather than a PDA and PC as in the case of this thesis project).

The time elapsed to redirect the secure media stream to the local node is around 1.3 seconds. When added with the time it takes to determine the location of the user (around 1.1 seconds), the overall delay is noticeable by the user. We believe this delay is acceptable for a small home and office environment where the number of users and devices is limited. This is especially true when one considers that a typical walking speed for a user is ~2m/s. However, in order to use this framework in a large scale deployment, a faster means of redirecting media streams should be investigated.

7. Conclusion and Future Work

7.1 Goal Attainment

The challenge of utilizing a user’s context to dynamically discover and use multimedia services via a secure SIP user agent was the core problem to be addressed in this project. Existing literature indicated that by utilizing simple sensors (such as passive infrared sensors), applications that detect meetings and determine occupancy information of a room can be developed [3; 32]. We developed and evaluated a new context aggregation framework that utilizes existing and new methods of sensing user’s current location. The framework gathers and analyzes context information from passive infrared sensors, infrared beacons, and light intensity & temperature sensors in order to determine the most probable location of the user.

The context aggregation framework has two important parts: a *room locator client* and *room locator server*. The client is an application installed on the user’s mobile device which initiates room location requests. A typical location request is an XML encoded message that includes infrared beacon numbers, light intensity and temperature readings in the user’s current location. The room locator server is a web application that receives the XML encoded location request. Based on the information provided in this location request, the server determines the location (in our case current room number) and action of the user (either entering or exiting). Once the location of the user is determined, our mobile SIP user agent is able to utilize this information to search for services available in this specific room.

For this purpose we have developed an SLP user agent that is capable of dynamically discovering multimedia devices such as projectors, speakers, cameras. The room locator client and the SLP user agent applications are integrated in to an existing SIP user agent called *Minisip* [19]. The resulting SIP user agent is able to dynamically discover and interact with nearby multimedia devices and services ***without*** requiring any manual configuration from the user.

This thesis project also addressed the challenge of establishing trust between the SIP user agent and the multimedia services. The main objective of establishing trust is to make sure that our SIP user agent does not start redirecting media streams to rogue services. The mechanism we proposed establishes a security context by authenticating service advertisements from the service providers. The established security context enables trusted services to participate in a secure multimedia session.

7.2 Conclusion

The accuracy measurements we have performed indicate that the context aggregation framework provides more accurate location determination when additional sensors are incorporated. By including infrared beacons, passive infrared sensors, and light intensity & temperature sensors, we have been able to obtain location accuracy greater than 80%. In comparison to the room occupancy sensor system [32], which only utilizes room status information published by a passive infrared sensor, our context aggregation framework provides an increase in accuracy of 20%-30%. Therefore, instead of relying on a single sensor, more accurate location determination can be achieved by utilizing context information gathered from multiple sources. Furthermore, we have shown that a centralized approach of aggregating and analyzing multiple sources provides even better accuracy.

However, adding more sensors and introducing more detailed analysis of context information causes additional delay. In order to determine if the latency introduced by including context-awareness to our SIP user agent is acceptable, we conducted performance tests. The results indicate that it takes about 1.1 seconds for the room locator client to determine the location of the user. Once the location of the user is determined, an additional 1.3 seconds is required to redirect the media stream. Despite the fact that a delay greater than one second is clearly noticeable by the user, we argue that due to the added functionality the user will find the increased delay is acceptable. The next section presents some suggestions that could help reduce this added delay and presents few open issues to be explored in future projects.

7.3 Future Work

It is important to note that the performance measurements we have conducted are in a laboratory with two target rooms and one mobile user. As the number of mobile users and the target locations increase, we anticipate that the context aggregation framework will exhibit increased latency. We believe the major portion of this increased delay is caused by the time spent to poll the light intensity and temperature sensors installed in all the rooms. The readings from these sensors are accessed via a serial port, which means in the current version of the framework only one request can be handled at a time. One possible solution could be implementing a separate process to access the light intensity and temperate readings of various rooms and make this data available to the room locator server. Such a process could continuously access the sensors and transmit the latest readings to the room locator server using UDP. Such a layered approach of implementing sensor systems is presented in Xueliang Ren's thesis [32] and could be incorporated into the context aggregation framework in the future. In addition to this the room locator client can be improved by parallelizing some of the operations. More specifically we have shown that by simultaneously accessing the light and temperature sensors while processing location requests could reduce the location determination delay by 400 ms making the framework more scalable. However, the remaining delay of ~450 ms might be further reduced.

Another future improvement is related to public key distribution. We have used a public key infrastructure in order to establish trust and create a security context between the mobile device and multimedia services. In the current version of the framework, in order to authenticate service advertisements, the mobile node requires access to public key files of all the services configured locally. On the multimedia services side, the client requires the public key of the mobile node in order to verify the S/MIME signature of the SDP that carries the SRTP keying material. This configuration of course works well for a small number of devices and services. However for wider use, requiring each device to manage all of the various public keys does not scale. For this purpose one could introduce a public key server to distribute public keys and the appropriate credentials. One possibility could be using an HTTP

Keyserver Protocol (HKP) [53] server to allow applications search and download public keys after network administrators have uploaded public keys using a web based interface.

Currently we used the command line based user interface of Minisip. In the future one could include a graphical user interface. There is some existing graphical user interface implementation using C #.Net for Pocket PC [19]. However, this user interface has no event handlers used to interact with the SIP and media layer. In order to implement event handlers, one could take advantage of the three layered approach used in Minisip. Minisip uses message based communication between the user interface, the SIP, and the media layer. In order to access some of the methods implemented in Minisip (using C++) from a managed code (such as C#.Net) the Platform Invoke (P/Invoke) technology [54] could be used.

Finally we would like to point out that current version of Minisip used on the iPAQ PDA running Pocket PC does not have support for video. The Linux version of Minisip has experimental support for video using H263 codec from the FFMPEG's *libavcodec* library [55]. Due to time constraints we did not cross compile this library for Windows CE platform. Future projects could port the *libavcodec* library to Windows CE platform to evaluate the context aggregation framework with video support.

References

1. **J. Rosenberg, H. Schulzrinne, et al.** *SIP: Session Initiation Protocol.* IETF, RFC 3261, June 2002.
2. **H. Sinnreich, A. B. Johnston.** *Internet Communications Using SIP: Delivering VoIP and Multimedia Services with Session Initiation Protocol. Second Edition.* New York : Wiley, 2006. 978-0-471-77657-4.
3. **Daniel Hübinette.** *Occupancy Sensor System, Master Thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication Technology, 2007. COS/CCS 2007-26, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/071221-Daniel_Hubinette_Master_Thesis-with-cover.pdf.
4. **Oscar Santillana.** *RTP redirection using a handheld device with Minisip, Master Thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication, 2007. COS/CCS 2007-10, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/070301-Oscar_Santillana-FinalVersion-with-cover.pdf.
5. **Shasha Zhang.** *Device aggregation with data networking, Master Thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication, 2008. COS/CCS 2008-23, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/081003-Shasha_Zhang-with-cover.pdf.
6. **Haruumi Shiode.** *In-building Location Sensing Based on WLAN Signal Strength: Realizing a Presence User Agent, Masters Thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication, 2008. COS/CCS 2008-04, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080314-Haruumi_Shiode-with-cover.pdf.
7. **Hu Lidan.** *An Intelligent Presentation System, Master Thesis* . Stockholm : Royal Institute of Technology (KTH), School of Information and Communication, 2008. COS/CCS 2008-14, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/080807-Hu_Lidan-with-cover.pdf.
8. **E. Guttman, C. Perkins, et al.** *Service Location Protocol, Version 2.* IETF, RFC 2608, June 1999.
9. **M. T. Smith and G. Q. Maguire Jr.** SmartBadge version 4. [Online] HP Labs and Royal Institute of Technology (KTH), January 03, 2001. [Cited: June 02, 2009.] http://web.it.kth.se/~maguire/badge4.html.
10. **W. H. Winsborough, K. E. Seamons, V. E. Jones.** *Automated Trust Negotiation.* South Carolina : In DARPA Information Survivability Conference and Exposition, 2000. ISBN:0769504906.
11. **A. K. Dey, G. D. Abowd.** *Towards a Better Understanding of Context and Context Awareness.* Karlsruhe : Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, 1999. ISBN:3-540-66550-1.
12. **C. Bolchini, C. A. Curino and et al,.** *A data-oriented survey of context models.* New York : ACM, 2007. ISSN:0163-5808.
13. UPnP Device Architecture 1.0. *UPnP Forum.* [Online] 24 April 2008. [Cited: 05 January 2009.] http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf.
14. Apache River Incubation Project. [Online] 24 April 2008. [Cited: 05 June 2009.] http://incubator.apache.org/river/RIVER/index.html.

15. **H. Chen, A. Joshi, and T. Finin.** *Dynamic Service Discovery for Mobile Computing: Intelligent Agents Meet Jini in the Aether.* Hingham : Kluwer Academic Publishers, 2001. ISSN:1386-7857.

16. **Gerald Q. Maguire Jr.** Practical Voice Over IP (VoIP): SIP and related protocols course(IK2554). [Online] Royal Institute of Technology (KTH), School of Information and Communication,, 12 January 2009. [Cited: 02 January 2009.] http://www.it.kth.se/courses/IK2554/VoIP-2008.pdf.

17. **H. Schulzrinne, S. Casner, et al.** *RTP: A Transport Protocol for Real-Time Applications.* IETF, RFC 3550, July 2003.

18. **Xiaokun Yi.** *Adaptive Wireless Multimedia Services, Masters thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication Technology, May 2006. COS/CCS 2006-12, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/060511-Xiaokun_Yi-Thesis-with-cover.pdf.

19. MiniSIP. [Online] 05 February 2009. [Cited: 05 February 2009.] http://www.minisip.org.

20. SIP Express Router. [Online] 01 January 2007. [Cited: 08 January 2009.] http://www.iptel.org/ser.

21. **M. Baugher, D. McGrew, et al.** *The Secure Real-time Transport Protocol (SRTP).* IETF, RFC 3711, March 2004.

22. **J. Arkko, E. Carrara, and et al.** *MIKEY: Multimedia Internet KEYing.* IETF, RFC 3830, August 2004.

23. **F. Andreasen, M. Baugher, and et al.** *Session Description Protocol (SDP) Security Descriptions for Media Streams.* IETF, RFC 4568, July 2006.

24. TrustBuilder. [Online] 01 January 2008. [Cited: 08 January 2009.] http://dais.cs.uiuc.edu/dais/security/trustb.php.

25. **H. Sugano, S. Fujimoto, et al.** *Presence Information Data Format (PIDF).* . : IEFT, August 2004. RFC 3863.

26. **J. Rosenberg.** *The Extensible Markup Language (XML) Configuration Access Protocol (XCAP).* IETF, RFC 4827, May 2007.

27. **David Sabaté Mogica.** *Remote Desktop: Integrating multiple devices.* Stockholm : Royal Institute of Technology(KTH), School of Information and Communications Technology, 2008. COS/CCS 2008-15, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/081202-DavidSabateMogica-report-with-cover.pdf.

28. OpenSLP. [Online] May 02, 2009. [Cited: May 02, 2009.] http://www.openslp.org/.

29. **S. Goyal, J. Carter.** *A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices.* Washington, DC : Mobile Computing Systems and Applications, 2004. WMCSA 2004. Sixth IEEE Workshop, 2004. ISBN: 0-7695-2258-0.

30. **Johan Sverin.** *Speech Interface for a Mobile Audio Application, Master Thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication,, 2005. IMIT/LCN 2005-17, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/050702-Johan_Sverin-with-cover.pdf.

31. Velleman Components. *PIR Intrusion Detector.* [Online] Velleman, January 02, 2009. [Cited: January 04, 2009.] http://www.velleman.be/be/en/product/view/?id=374758.

32. **Xueliang Ren.** *A Meeting Detector to Provide Context to a SIP Proxy, Masters thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication, October 2008. COS/CCS 2008-24, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/081025-Xueliang_Ren-with-cover.pdf.

33. **Ke Wang.** *Exploiting Presence.* Stokholm : Royal Institute of Technology (KTH), School of Information and Communication, December 2008. COS/CCS 2008-27, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/081205-Ke_Wang-with-cover.pdf.

34. **Alisa Devlic.** *Context-Addressed Communication Dispatch, Licentiate Thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication, May 2009. http://web.it.kth.se/~devlic/licentiate%20thesis/Alisa_Devlic-licentiate-thesis.pdf.

35. **J. Rosenberg, J. Peterson, et al.** *Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP).* IETF, RFC 3725, April 2004.

36. **Wesley Chan.** *Using CoolBase to Build Ubiquitous Computing Applications.* 2001.

37. **California Wireless, Inc., Ericsson, et al.** *Infrared Data Association Guidlines for Ultra Protocols.* . : IrDA, October 1997. V.1.

38. **Mark T. Smith.** Course website for II2302: Sensor Based Systems. [Online] Royal Institute of Technology (KTH), School of Information and Communication,, January 1, 2009. [Cited: June 29, 2009.] http://web.ict.kth.se/~msmith/II2302_2009.html.

39. FTD Chip VCP Driver. [Online] May 12, 2008. [Cited: June 12, 2009.] http://www.ftdichip.com/Drivers/VCP.htm.

40. MSDN forum discussion on bug when using IO.Port.SerialPort class to access emmulated IrDA port. *Title of discussion: "IO.Ports.SerialPort - issue with .BytesToRead and DataReceived Event".* [Online] January 01, 2009. [Cited: June 14, 2009.] http://social.msdn.microsoft.com/Forums/en-US/netfxcompact/thread/1c548a4d-2556-4ee4-9f01-b0922fd96862/.

41. OpenNetCF Consulting. [Online] June 13, 2008. [Cited: June 15, 2009.] http://www.opennetcf.com/.

42. FUJI & CO.(Piezo Science) MPY-20C48 LDR. [Online] May 31, 2009. [Cited: June 13, 2009.] http://www.fuji-piezo.com/photoldr.htm.

43. **Joachim Köppen.** Photometry with Light Dependent Resistors. [Online] 01 May 2004. [Cited: 22 May 2009.] http://www.astrophysik.uni-kiel.de/~koeppen/blueskies/photometer.html.

44. JAIN-SIP API. [Online] January 01, 2007. [Cited: June 13, 2009.] https://jain-sip.dev.java.net/.

45. **Mikael Svensson.** *Countering VoIP Spam: Up-Cross-Down Certificate Validation.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication,, September 2007. http://www.minisip.org/publications/Thesis_Svensson_Sep2007.pdf.

46. *NIST Digitalsignature Standard Technical Report.* : U.S. Department of Commerce, May 1994. NIST FIPS PUB 186.

47. Cryptographic Interoperability: Digital Signatures. *The code project.* [Online] 26 April 2008. [Cited: 02 April 2009.] http://www.codeproject.com/KB/security/CryptoInteropSign.aspx.

48. **J. Peterson.** *S/MIME Advanced Encryption Standard (AES) Requirement for the Session Initiation Protocol (SIP).* IETF, RFC 3853, July 2004.

49. **Thor Hådén's.** *IPv6 Home Automation, Batchelors thesis.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication, June 2009. TRITA-ICT-EX-2009:28, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/090601-Thor_Haaden.pdf.

50. Wireshark. [Online] January 01, 2009. [Cited: June 20, 2009.] http://www.wireshark.org.

51. SIP Scenario Generator. [Online] 25 July 2004. [Cited: 22 June 2009.] http://www.iptel.org/~sipsc/.

52. **Johan Bilien, Erik Eliasson, et al.** Secure VoIP: call establishment and media protection. [Online] 01 January 2009. [Cited: 12 June 2009.] http://www.minisip.org/publications/secvoip-minisip-camera.pdf.

53. **D. Shaw.** *The OpenPGP HTTP Keyserver Protocol (HKP).* IETF, March 2003. http://tools.ietf.org/html/draft-shaw-openpgp-hkp-00.

54. Platform Invoke (P/Invoke). [Online] January 01, 2009. [Cited: June 22, 2009.] http://msdn.microsoft.com/en-us/library/aa288468(VS.71).aspx.

55. FFMPEG.ORG. [Online] May 01, 2008. [Cited: June 12, 2009.] http://ffmpeg.org/index.html.

56. **Athanasios Karapantelakis.** *A mobile SIP client: From the user interface design to evaluation of synchronised playout from multiple SIP user agents.* Stockholm : Royal Institute of Technology (KTH), School of Information and Communication,, 2007. COS/CCS 2007-07, http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/070227-Athanasios_Karapantelakis-thesis_final-with-cover.pdf.

Appendices

A. Useful tools used when developing application for Pocket PC

Name	Description	Source
PHM Device Manager	The Device Manager display a graphical view of the hardware built-in or connected to your Pocket PC.	http://www.phm.lu/products/PocketPC/DevMgmt/
Windows Mobile Developer Power Toys	Pocket PC command shell, remote Pocket PC display and others.	http://www.microsoft.com/downloads/details.aspx?FamilyID=74473fd6-1dcc-47aa-ab28-6a2b006edfe9&displaylang=en
PocketPuTTY	Terminal emulator for Pocket PC. Supports SSH, Telnet, and Serial. Useful when debugging the Wasa Board.	http://www.pocketputty.net/
PocketPC Ping	A utility application used to ping IP or host names.	http://sourceforge.net/projects/pocketping/
MyIpConfig	Shows current IP configuration for multiple network adapters	http://sourceforge.net/projects/myipconfig/

B. Connecting a Wasa board to the HP iPAQ PDA

The following steps provide the tasks required to install the Wasa board on the iPAQ in order to access the light intensity and temperature readings.

1. The Wasa board uses a USB interface to access the ADC readings, therefore we must first install USB host on the iPAQ.
2. Insert the iPAQ in to the expansion pack as shown in the figure below.
3. Insert the USB host card in to the expansion card. We have used the SolarExpress PDA II CF USB Host.
4. Install the USB host driver. We have used the one the comes with the card but it is also available at http://www.lightconecorp.net/blog.
5. Soft reset the PDA.
6. Connect the Wasa board with the PDA as shown in the figure.
7. Install the Virtual Com Port (VCP) driver for the Wasa board. Download the correct version from http://www.ftdichip.com/Drivers/VCP.htm.
8. Copy the ftdi_ser.dll and FTDIPORT.INF files to the \\Windows directory of the PDA.
9. Reset the device and when it asks you for the name of the device driver enter ftdi_ser.dll so that the appropriate driver could be located.
10. Now the Wasa board is installed. You can check your installation using PocketPuTTY.

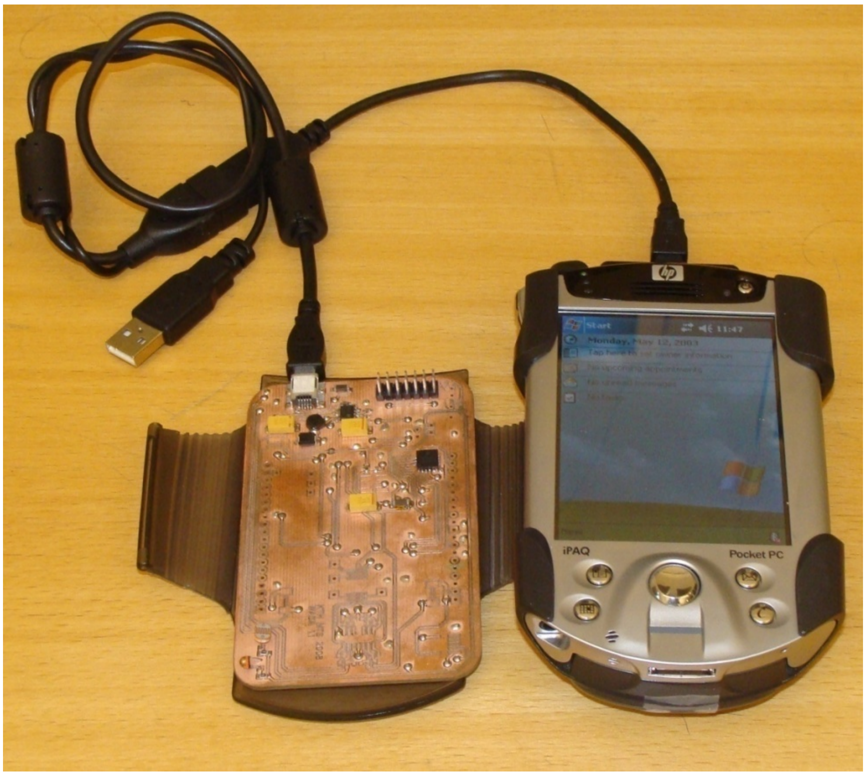


Figure 41: Connecting the Wasa board to the HP iPAQ PDA

C. Partial listing of the IRBeaconReader class.

This is partial listing of the IRBeaconReader class used on the iPAQ PDA to read the URL broadcasted by the beacon. The part of the code that create the XML location request is removed and the full source code can be checked out from my SVN server by issuing the following command:

```
svn checkout http://securecontextawaresip.googlecode.com/svn/trunk/
securecontextawaresip-read-only
```

```
namespace minisip.context{
    class IRBeaconReader{
        private Port irPort;
        private StringBuilder irData;
        /// <summary>
        /// Configure the serial port
        /// </summary>
        public IRBeaconReader(){
            irPort = new Port("COM2:");
            irPort.RThreshold = 1;
            irPort.Settings.BaudRate = BaudRates.CBR_9600;
            irPort.Settings.ByteSize = 8;
            irPort.Settings.Parity = Parity.none;
            irPort.Settings.StopBits = StopBits.one;
            irPort.IREnable = true; //set to true to indicate that this is emulated
IR port.
            irData = new StringBuilder();
        }
        /// <summary>
        /// Setup receiver event and error handlers.
        /// </summary>
        public void startReading(){
            try{
                irPort.OnError += new Port.CommErrorEvent(onError);
                irPort.DataReceived += new Port.CommEvent(onData);
                bool sucess = irPort.Open(); //Open the IR Port
                System.Console.WriteLine("IR-Port Opened : " + sucess);
            }
            catch (Exception ex){
                System.Console.WriteLine(ex.ToString());
                irPort.Close();
            }
        }

        /// <summary>
        /// Closes the IR port.
        /// </summary>
        public void stopReading(){
            irPort.Close();
            System.Console.WriteLine("Closed IR port successfully.");
        }

        /// <summary>
        /// This deligate will be called when read error occurs
        /// </summary>
        /// <param name="d">The error message.</param>
        public void onError(string d){
            System.Console.WriteLine("Error: " + d);
        }
        /// <summary>
        /// This deligate will be called when data is avaiable
        /// </summary>
        public void onData(){
            try{
                byte[] buff = irPort.Input; //Save the reciever buffer
                String data = new ASCIIEncoding().GetString(buff, 0, buff.Length);
                String startWith = "????????????";
                if (data.StartsWith(startWith)){
                    //read the first part of the url its something like this
"????????????0phttp://www.it.kt"
                    L20Start = DateTime.Now;
                    irData.Remove(0, irData.Length);
                    String url = data.Substring(data.IndexOf("http"));
                    irData.Append(url);
                }
                else //read the second part
                {
                    //trunkate the last 4 characters. It is something like this
"h.se/~maguire/beacon/1\0à?"
                    int len = data.Length - 4;
                    String url = data.Substring(0, len);
                    irData.Append(url);
                    System.Console.WriteLine("URL: " + irData.ToString());
                    String beacon = url.Substring(url.IndexOf("beacon"));
                    //Now we have extracted the URL and the beacon number. Continue
creating location request XML
                    //...
                }
            }
            catch (Exception ex){
                System.Console.WriteLine(ex);
                fileWriter.Close();
                Console.ReadLine();
                Process.GetCurrentProcess().Kill();
            }
        }
    }
}
```

D. Partial listing of the WasaBoardReader class

This is partial listing of WasaBoardReader class used by the iPAQ PDA to access the light and temperature sensors on the Wasa board. The full source code can be checked out from my SVN server by issuing the following command:

```
    svn checkout http://securecontextawaresip.googlecode.com/svn/trunk/
securecontextawaresip-read-only
```

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO.Ports;
using System.Threading;

namespace light_temp_senor_test
{
    public class WasaSensorReader
    {
        public static String LIGHT_SENSOR = "LIGHT_SENSOR";
        public static String TEMP_SENSOR = "TEMP_SENSOR";

        private String SENSOR_COM_PORT = "COM0";
        private static int READ_TIMEOUT = 10000;
        private static int MAX_LIGHT = 4096; //Since we have 12bit ADC.
        private static int MAX_TEMP = 4096;//Since we have 12bit ADC.
        private SerialPort serial;
        /// <summary>
        /// Creates a COM port for the given port number
        /// </summary>
        /// <param name="comPort">Port number</param>
        public WasaSensorReader(String comPort)
        {
            this.SENSOR_COM_PORT = comPort;
            serial = new SerialPort(SENSOR_COM_PORT, 115200, Parity.None, 8,
StopBits.One);
            serial.Handshake = Handshake.None;
            serial.Open();
            System.Console.WriteLine("WasaSensorReader: Opened port " +
SENSOR_COM_PORT);
            serial.ReadTimeout = READ_TIMEOUT;
        }

        /// <summary>
        /// Connects to the Wasa board and read numberOfReadings times
        /// </summary>
        /// <param name="sensorType">Type of reading "LIGHT_SENSOR" or
"TEMP_SENSOR" </param>
        /// <param name="numberOfReadings">Number consucative of readings.</param>
        /// <returns>The mean of the readings.</returns>
        public long readValue(String sensorType, int numberOfReadings)
        {

            long sum = 0;
            for (int i = 0; i < numberOfReadings; i++)
            {
                //issue the approprate AT command
                if (sensorType.Equals(LIGHT_SENSOR))
                {
                    /*The LDR is connected to analog input 6
                    hence the s-register AT command will be AT S206?\r*/
                    serial.Write("AT S206?\r");
                }
                else
                {
                    /*The TDR is connected to analog input 5,
                    hence the s-register AT command will be AT S205?\r*/
                    serial.Write("AT S205?\r");
                }
                try
                {
                    byte[] b = new byte[serial.ReadBufferSize];
                    int n = serial.Read(b, 0, serial.ReadBufferSize);//read the
input buffer.
                    String data = new ASCIIEncoding().GetString(b, 0, n);
                    data = data.Replace("OK", "");//remove the OK output if echo is
set to true.
                    data = data.Replace("\n", "");//remove new line characters
                    data = data.Replace("\r", "");
                    data = data.Trim();
                    System.Console.WriteLine(sensorType + "[" + i + "]" + data);
                    long reading = 0;
                    try
                    {
                        if (!data.Equals(""))
                        {
                            reading = long.Parse(data);
                        }
                    }
                    catch (FormatException ex)
                    {
                        System.Console.WriteLine(ex);

                    }
                    if (sensorType.Equals(LIGHT_SENSOR))
                    {
                        if (reading > MAX_LIGHT || reading <= 0)
                        {
                            //if the reading is not valid read one more time
                            i--;
                            continue;
                        }
                    }
                    else
                    {
                        if (reading > MAX_TEMP || reading <= 0)
                        {
                            //if the reading is not valid read one more time
                            i--;
                            continue;
                        }
                    }
                    sum += reading;
                }
                catch (TimeoutException e)
                {
                    System.Console.WriteLine(e);
                    return 0;
                }
            }
            return (sum / numberOfReadings);
        }
        public void closePort()
        {
            serial.Close();
        }
    }
}
```

E. Partial listing of the RoomSubscriber class

This class is used to implement a SIP presence watcher that provides additional contextual information for the room locator service. When status of a room changes (i.e when someone enters or leaves the room), the SIP proxy will send us a NOTIFY message and the information received will be stored in a database. The room locator server will use this information when making decision about the location of the user. The full source code can be checked out from my SVN server by issuing the following command:

```
     svn checkout http://securecontextawaresip.googlecode.com/svn/trunk/
securecontextawaresip-read-only
```

```
public class RoomWatcher implements SipListener{
       private static final String PROXY_IP = "192.168.2.238";//"130.237.15.238";
       private static final int PROXY_PORT = 5060;
       private static final String PROXY_TRANSPORT = "udp";

       private static final String MY_IP =  "192.168.2.90";//"130.237.239.208";
       private static final int MY_PORT = 5060;
       private static final String MY_TRANSPORT = "udp";

       private static SipProvider sipProvider;
       private static AddressFactory addressFactory;
       private static MessageFactory messageFactory;
       private static HeaderFactory headerFactory;
       private static SipStack sipStack;
       private ContactHeader contactHeader;

       protected ClientTransaction subscribeTid;

       public void processRequest(RequestEvent requestReceivedEvent) {
              Request request = requestReceivedEvent.getRequest();
              ServerTransaction serverTransactionId =
                     requestReceivedEvent.getServerTransaction();
              if (request.getMethod().equals(Request.NOTIFY)){
                     processNotify(request, serverTransactionId);
              }
              else{
                     System.out.println("Watcher Error: Unsupported method: "+
                                   request.getMethod());
              }
       }

       private void processNotify(Request request, ServerTransaction
serverTransactionId) {
       try {
              if (serverTransactionId == null) {
                     System.out.println("Watcher Error: Got null server transaction
ID.");
                     return;
              }
              Dialog dialog = serverTransactionId.getDialog();
              Response response = messageFactory.createResponse(200,request);
              serverTransactionId.sendResponse(response);
              SubscriptionStateHeader subscriptionState =

       (SubscriptionStateHeader)request.getHeader(SubscriptionStateHeader.NAME);
              //Check if the subscription is terminated.
              if (
subscriptionState.getState().equals(SubscriptionStateHeader.TERMINATED)) {
                     dialog.delete();
                     System.out.println("Watcher Info: subscription terminated.");
              }
              //update the room database with the the new location update
              updatePIRState(new String(request.getRawContent()));
```

```
    } catch (Exception ex) {
        ex.printStackTrace();
        System.exit(0);
    }
    }

    public void processResponse(ResponseEvent responseReceivedEvent) {
        Response response = (Response) responseReceivedEvent.getResponse();
        //System.out.println("Watcher Info: Got a
response.\n"+response.toString());
        Transaction tid = responseReceivedEvent.getClientTransaction();

        System.out.println("Watcher Info: Response received "+ response);
    }

    public void init() {
        SipFactory sipFactory = null;
        sipFactory = SipFactory.getInstance();
        sipFactory.setPathName("gov.nist");
        Properties properties = new Properties();
        //setup the sip stack properties.
        properties.setProperty("javax.sip.OUTBOUND_PROXY",
        PROXY_IP+":"+Integer.toString(PROXY_PORT)+"/" + PROXY_TRANSPORT);
        properties.setProperty("javax.sip.STACK_NAME", "subscriber");
        properties.setProperty("javax.sip.MAX_MESSAGE_SIZE", "1048576");
    properties.setProperty("gov.nist.javax.sip.DEBUG_LOG","subscriberdebug.txt");

    properties.setProperty("gov.nist.javax.sip.SERVER_LOG","subscriberlog.txt");
        properties.setProperty("gov.nist.javax.sip.CACHE_CLIENT_CONNECTIONS",
"false");
        properties.setProperty("gov.nist.javax.sip.TRACE_LEVEL", "32");

        try {
            // Create SipStack object
            sipStack = sipFactory.createSipStack(properties);
        } catch (PeerUnavailableException e) {
            e.printStackTrace();
            System.exit(0);
        }

        //Create and send a SUBSCRIBE request to proxy
        try {
            headerFactory = sipFactory.createHeaderFactory();
            addressFactory = sipFactory.createAddressFactory();
            messageFactory = sipFactory.createMessageFactory();

            ListeningPoint lp =
                sipStack.createListeningPoint(MY_IP,MY_PORT,
MY_TRANSPORT);
            sipProvider = sipStack.createSipProvider(lp);
            sipProvider.addSipListener(this);

            String fromName = "room_locator";
            String fromSipAddress = PROXY_IP;
            String fromDisplayName = "Room Locator Service";
            String toSipAddress = PROXY_IP;
            String toUser = "pir";
            String toDisplayName = "PIR Sensor";

            // create >From Header
            SipURI fromAddress = addressFactory.createSipURI(fromName,
fromSipAddress);
            Address fromNameAddress =
addressFactory.createAddress(fromAddress);
            fromNameAddress.setDisplayName(fromDisplayName);
            FromHeader fromHeader =
                headerFactory.createFromHeader(fromNameAddress, "12345");
```

```
                    // create To Header
                    SipURI toAddress = addressFactory.createSipURI(toUser,
toSipAddress);
                    Address toNameAddress =
addressFactory.createAddress(toAddress);
                    toNameAddress.setDisplayName(toDisplayName);
                    ToHeader toHeader = headerFactory.createToHeader(toNameAddress,
null);
                    // create Request URI
                    SipURI requestURI = addressFactory.createSipURI(toUser,
toSipAddress);
                    // Create ViaHeaders
                    ArrayList viaHeaders = new ArrayList();
                    ViaHeader viaHeader =
                           headerFactory.createViaHeader(MY_IP, MY_PORT,
MY_TRANSPORT, null);
                    // add via headers
                    viaHeaders.add(viaHeader);
                    // Create a new CallId header
                    CallIdHeader callIdHeader = sipProvider.getNewCallId();
                    // Create a new Cseq header
                    CSeqHeader cSeqHeader = headerFactory.createCSeqHeader(1L,
Request.SUBSCRIBE);
                    // Create a new MaxForwardsHeader
                    MaxForwardsHeader maxForwards =
headerFactory.createMaxForwardsHeader(70);
                    // Create the request.
                    Request request =
                           messageFactory.createRequest(requestURI,
Request.SUBSCRIBE, callIdHeader,
                                 cSeqHeader,fromHeader,     toHeader, viaHeaders,
maxForwards);
                    // Create contact headers
                    String host = lp.getIPAddress();
                    SipURI contactUrl = addressFactory.createSipURI(fromName,
host);
                    contactUrl.setPort(lp.getPort());
                    // Create the contact name address.
                    SipURI contactURI = addressFactory.createSipURI(fromName,
host);

       contactURI.setPort(sipProvider.getListeningPoint(MY_TRANSPORT).getPort());
                    Address contactAddress =
addressFactory.createAddress(contactURI);
                    // Add the contact address.
                    contactAddress.setDisplayName(fromName);
                    contactHeader =
headerFactory.createContactHeader(contactAddress);
                    request.addHeader(contactHeader);
                    //add event header.
                    EventHeader eventHeader =
headerFactory.createEventHeader("presence");
                    request.addHeader(eventHeader);
                    AcceptHeader acceptHeader =
                           headerFactory.createAcceptHeader("application",
"pidf+xml");
                    request.addHeader(acceptHeader);
                    // Create the client transaction.
                    this.subscribeTid =
sipProvider.getNewClientTransaction(request);
                    // send the request out.
                    subscribeTid.sendRequest();
                    System.out.println("Watcher Info: Subscribe
sent.\n"+request.toString());

             } catch (Exception ex) {
                    ex.printStackTrace();
                    System.exit(1);
```

```
        }
    }

    //used to update the database with the room location update from the PIDF XML
    private void  updatePIRState(String xml){
    try{
        DocumentBuilderFactory docFactory = 
DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = docFactory.newDocumentBuilder();
        Reader reader = new StringReader(xml);
        Document doc = builder.parse(new InputSource(reader));
        NodeList child = doc.getChildNodes();
        //get action tag value (action can be exit or enter)
        NodeList nodes = doc.getElementsByTagName("action");
        String action = null;
        if(nodes.getLength()!=0){
            Element e = (Element)nodes.item(0);
            Node n = e.getFirstChild();
            CharacterData data = null;

            if(n instanceof CharacterData){
                data = (CharacterData)n;
                System.err.println("Action node: "+data.getData());
                action = data.getData();
            }
        }
        //get area (name of the room)
        nodes = doc.getElementsByTagName("area"); //we only have one action 
element
        String area = null;
        if(nodes.getLength()!=0){
            Element e = (Element)nodes.item(0);
            Node n = e.getFirstChild();
            CharacterData data = null;
            if(n instanceof CharacterData){
                data = (CharacterData)n;
                System.err.println("Area node: "+data.getData());
                area = data.getData();
            }
        }
        //Open JDBC connection to database server.
        if(area!=null && action !=null){
            String url = 
"jdbc:sqlserver://localhost:1433;databaseName=room_info;";
            Connection con;

            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            con = java.sql.DriverManager.getConnection(url,"sa","pass4u");
            if(con!=null){
                System.out.println("Database Connection Successful!");
                Statement updateStmt = con.createStatement();
                String sql="";
                //update the last_enterance or last_exit field of
                //the room table with the current time
                if(action.equalsIgnoreCase("entry")){
                    sql = "update room set last_enterance =getDate() 
where name='"+area+"';";
                }
                else if(action.equalsIgnoreCase("exit")){
                    sql = "update room set last_exit =getDate() where 
name='"+area+"';";
                }

                int affect = updateStmt.executeUpdate(sql);
                System.out.println("Affected rows"+affect);

            }
            else{
```

```
                           System.out.println("Error creating database
connection.");
                           }
                    }
             }
             catch(Exception ex)
             {
                    ex.printStackTrace();
             }
       }

       public static void main(String[] arg){
             new Subscriber().init();
       }

       //....
       //Override other methods of the SipListener class
       //....
     }
```

F.Partial listing of the room locator client

This is portion of the IRBeaconReader class that implements two method createRequest() and uploadRequest() for implementing the room locator client functionality.

```
namespace minisip.context{
    class IRBeaconReader{
        private Port irPort;
        private StringBuilder irData;
        /// <summary>
        /// Configure the serial port
        /// </summary>
        public IRBeaconReader(){
        }
        /// <summary>
        /// Setup receiver event and error handlers.
        /// </summary>
        public void startReading(){
        }
        /// <summary>
        /// Closes the IR port.
        /// </summary>
        public void stopReading(){
        }
        /// <summary>
        /// This deligate will be called when read error occurs
        /// </summary>
        /// <param name="d">The error message.</param>
        public void onError(string d){
        }
        /// <summary>
        /// This deligate will be called when data is avaiable
        /// </summary>
        public void onData(){
        }
        /// <summary>
        /// Creates XML file that contains the temprature and light
readings
        /// </summary>
        /// <param name="beaconNumber">the beacon number to use in the
request</param>
        /// <returns>XML of the location request</returns>
        private String createRequest(int beaconNumber)
        {
            if (null != xmlDoc)
            {
                //Set the Beacon number
                XmlNodeList beaconReadings =
xmlDoc.GetElementsByTagName("beacon");
                beaconReadings[0].InnerText = beaconNumber.ToString();

                //Set the Temprature value

                WasaSensorReader wasaReader = new WasaSensorReader("COM0");
                //read 5 temp values
                long tempMean =
wasaReader.readValue(WasaSensorReader.TEMP_SENSOR, 1);
                XmlNodeList tempReadings =
xmlDoc.GetElementsByTagName("temp");
                tempReadings[0].InnerText = tempMean.ToString();
                //Set the Light value.
                long lightMean =
wasaReader.readValue(WasaSensorReader.LIGHT_SENSOR, 1);
                XmlNodeList lightReadings =
xmlDoc.GetElementsByTagName("light");
                lightReadings[0].InnerText = lightMean.ToString();
                wasaReader.closePort();

                xmlDoc.Save("\\minisip\\request.xml");
                TimeSpan s = DateTime.Now.Subtract(L20Stop);
                fileWriter.WriteLine("L21(ms)=" + s.TotalMilliseconds);
                fileWriter.Flush();

            }
            return xmlDoc.InnerXml;
        }
        /// <summary>
        /// Uploads the request to the room location server
        /// </summary>
        /// <param name="roomLocatorURI">URI of the location server, i.e. a
web address</param>
        /// <param name="requestXml">the request xml</param>
        /// <returns>Possible locations</returns>
        private LocationReply[] uploadRequest(String roomLocatorURI, String
requestXml)
        {

            byte[] reqBytes = new ASCIIEncoding().GetBytes(requestXml);
            //write file to uri
            HttpWebRequest request = HttpWebRequest.Create(roomLocatorURI)
as HttpWebRequest;
            request.Method = "POST";
            request.ContentType = "text/xml";
            request.ContentLength = reqBytes.Length;
            Stream reqStream = request.GetRequestStream();
            reqStream.Write(reqBytes, 0, reqBytes.Length);

            reqStream.Flush();
            reqStream.Close();
            System.Console.WriteLine("Location request sent to " +
roomLocatorURI);
            //read response

            HttpWebResponse response = request.GetResponse() as
HttpWebResponse;
            Stream respStream = response.GetResponseStream();

            if (response.ContentLength == 0)
            {
                throw new Exception("Error: Room locator reply is Empty.");
            }

            byte[] respData = new byte[response.ContentLength];
            respStream.Read(respData, 0, respData.Length);
            respStream.Close();
            System.Console.WriteLine("Reply Received from service.");
            //write response to file.

            BinaryWriter binWriter = new
BinaryWriter(File.Open("\\minisip\\response.xml",
                FileMode.Create, FileAccess.Write));
            binWriter.Write(respData, 0, respData.Length);
            binWriter.Close();
            System.Console.WriteLine("Reply written to file.");


            //Construct LocationReply and return
            XmlDocument replyXml = new XmlDocument();
            replyXml.LoadXml(new ASCIIEncoding().GetString(respData, 0, 
respData.Length));

            XmlNodeList roomNodes = replyXml.GetElementsByTagName("room");
            LocationReply[] locationReply = new 
LocationReply[roomNodes.Count];
            for (int i = 0; i < roomNodes.Count; i++)
            {
                locationReply[i] = new LocationReply();
                XmlAttributeCollection roomAttributes = 
roomNodes[i].Attributes;
                for (int j = 0; j < roomAttributes.Count; j++)
                {
                    switch (roomAttributes[j].Name)
                    {
                        case "id":
                            locationReply[i].roomID = 
int.Parse(roomAttributes[j].Value);
                            break;
                        case "name":
                            locationReply[i].roomName = 
roomAttributes[j].Value;
                            break;
                        case "action":
                            locationReply[i].action = 
roomAttributes[j].Value;
                            break;
                        case "certainty":
                            locationReply[i].certainity = 
float.Parse(roomAttributes[j].Value);
                            break;
                    }
                }
            }
            return locationReply;
        }
    }
}
```

G. Partial listing of the room locator server

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Text;
using System.IO;
using System.Xml;
using Room_Locator_Web_App.RoomDataSetTableAdapters;
using Room_Locator_Web_App;
using light_temp_senor_test;
namespace Room_Locator_Web_App
{
    public partial class FileUploader : System.Web.UI.Page
    {
        private const String ACTION_ENTERANCE = "enterance";
        private const String ACTION_EXIT = "exit";
        private const String ACTION_UNKNOWN = "unknown";
        protected void Page_Load(object sender, EventArgs e)
        {
            try
            {
                if (Request.ContentLength == 0)
                {
                    Response.Redirect("Registration.aspx");
                    log("Web App accessed from desktop browser.");
                    return;
                }
                //todo check content type...
                Stream reqInStream = Request.InputStream;
                byte[] reqBytes = new byte[Request.ContentLength];
                reqInStream.Read(reqBytes, 0, reqBytes.Length);
                String reqString = new ASCIIEncoding().GetString(reqBytes);
                log("Request from client:\n" + reqString);
                XmlDocument reqDoc = new XmlDocument();
                reqDoc.LoadXml(reqString);

                byte[] respBytes = getResponse(reqDoc);
                log("Response to Client:\n" + new
ASCIIEncoding().GetString(respBytes));
                if (respBytes != null)
                {
                    //write back the reply
                    Stream reqOutStream = Response.OutputStream;
                    reqOutStream.Write(respBytes, 0, respBytes.Length);
                }
            }
            catch (Exception ex)
            {
                log(ex.ToString());
            }
        }

        public byte[] getResponse(XmlDocument reqDoc)
        {
            //1. parse out beacon, light and temprature values.
            //2. Check if we have a room with the given beacon number
            //  2.1 select the recent action(entry or exit)
            //  2.2 create xml response to this room
            //3. if we don't have the beacon number
            //  3.1 select the room with the clothses light and temp values
            //  3.2 select the room recent action(entry/exit)
            //  3.2 create xml response to this room
            bool isValidReq = false;
            String reqID = "";
            XmlNode message = reqDoc.GetElementsByTagName("message")[0];
            XmlAttributeCollection msgAttributes = message.Attributes;
            for (int i = 0; i < msgAttributes.Count; i++)
            {
                XmlAttribute attribute = msgAttributes[i];
                if (attribute.Name.Equals("type") &&
attribute.Value.Equals("request"))
                {
                    isValidReq = true;
                }
                if (attribute.Name.Equals("id"))
                {
                    reqID = attribute.Value;
                }

            }
            if (isValidReq)
            {
                String beacon =
reqDoc.GetElementsByTagName("beacon")[0].InnerText;
                String userLight = "";
                userLight =
reqDoc.GetElementsByTagName("light")[0].InnerText;
                String userTemp = "";
                userTemp =
reqDoc.GetElementsByTagName("temp")[0].InnerText;
                String actionByPIR = "";
                String actionByLightSensor = "";
                String actionByTempSensor = "";
                String actionFinal = "";
                int roomId = 0;
                String roomName = "";
                //Find the matching room record
                RoomTableAdapter roomTableAdapter = new RoomTableAdapter();
                RoomDataSet.RoomDataTable roomTable =
roomTableAdapter.GetData();
                RoomDataSet.RoomRow[] roomRows =
(RoomDataSet.RoomRow[])roomTable.Select();

                foreach (RoomDataSet.RoomRow row in roomRows)
                {
                    //match the beacon id
                    if (row.ir_beacon.Equals(beacon))
                    {
                        roomId = row.id;
                        roomName = row.name;
                        //enterance is more recent
                        //check also if the last_enterance and exit values
are updated within the last X sec.
```

```
                        //X will be determined experimentall on how long it
takes for a publish to cause a notify.
                        //for now it will be set 10 seconds
                        int maxDelay = 10;
                        if (!row.Islast_enteranceNull() &&
!row.Islast_exitNull())
                        {
                            DateTime now = DateTime.Now;
                            TimeSpan enteranceDiff =
now.Subtract(row.last_enterance);
                            TimeSpan exitDiff =
now.Subtract(row.last_exit);
                            //if the last enterance field is still valid
and is more recent than the exit value
                            if (enteranceDiff.TotalSeconds > maxDelay &&
                                row.last_enterance.CompareTo(row.last_exit)
> 0)
                            {
                                actionByPIR = ACTION_ENTERANCE;
                            }
                            else if (exitDiff.TotalSeconds > maxDelay &&
                                row.last_exit.CompareTo(row.last_enterance)
> 0) //other wise for the exit field
                            {
                                actionByPIR = ACTION_EXIT;
                            }
                            else
                            {
                                actionByPIR = ACTION_UNKNOWN;
                            }
                        }
                        else
                        {
                            actionByPIR = ACTION_UNKNOWN;
                        }

                        //retrive the room temprature and light values.
                        WasaSensorReader reader = new
WasaSensorReader("COM16");

                        long roomLight =
reader.readValue(WasaSensorReader.LIGHT_SENSOR, 5);
                        long roomTemp =
reader.readValue(WasaSensorReader.TEMP_SENSOR, 5);

                        reader.closePort();

                        //the light and temp vaules will be used to
determine if the user is in the given room
                        //this will be used to support/correct the

                        if (!userLight.Equals(""))
                        {
                            //experimentally we have identified that the
light sensor values within one room
                            //have a maximum divation less than 61.96.
                            //light readings are mean of atleast 5
consucative readings...why 5?
                            long lightDiff = Math.Abs(roomLight -
long.Parse(userLight));
```

```
                            if (lightDiff < 62)
                            {
                                actionByLightSensor = ACTION_ENTERANCE;
                            }
                            else
                            {
                                actionByLightSensor = ACTION_EXIT;
                            }
                        }
                        else
                        {
                            actionByLightSensor = ACTION_UNKNOWN;
                        }

                        if (!userTemp.Equals(""))
                        {
                            //Determine the temprature difference between
rooms and use it here
                            //for now we assume 20
                            long tempDiff = Math.Abs(roomTemp -
long.Parse(userTemp));
                            if (tempDiff < 20)
                            {
                                actionByTempSensor = ACTION_ENTERANCE;
                            }
                            else
                            {
                                actionByTempSensor = ACTION_EXIT;
                            }
                        }
                        else
                        {
                            actionByTempSensor = ACTION_UNKNOWN;
                        }

                        //Determine the final action
                        //the priority is pir, light, temp from high to low

                        if (actionByPIR.Equals(ACTION_UNKNOWN))
                        {
                            if (actionByLightSensor.Equals(ACTION_UNKNOWN))
                            {
                                //tie breaker is action by temp...may be
unknown
                                actionFinal = actionByTempSensor;
                            }
                            else
                            {
                                actionFinal = actionByLightSensor;
                            }
                        }
                        else
                        {
                            actionFinal = actionByPIR;
                        }
                        //insert to history table
                        HistoryTableAdapter historyTableAdapter = new
HistoryTableAdapter();
                        historyTableAdapter.Insert(roomId,
int.Parse(beacon), row.last_enterance,
                            row.last_exit, int.Parse(userLight),
int.Parse(userTemp),
                            (int)roomLight, (int)roomTemp, actionByPIR,
actionByLightSensor,
                            actionByTempSensor, actionFinal);

                        //construct the XML
                        //Todo fix timestamp..for now using fixed value
                        //Todo fix certainity value.
                        String xml =
                        "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
                        "<message type=\"reply\" id=\"" + reqID + "\"
time_stamp=\"225451787785\">" +
                        "<room id=\"" + roomId + "\" name=\"" + roomName +
"\" action= \"" +
                        actionFinal + "\" certainty=\"0.73\"/>" +
                        "</message>";
                        return new ASCIIEncoding().GetBytes(xml);
                    }
                }

            }
            return null;
        }
    }
}
```

H. Partial listing of the mobile SLP User agent

```
class SLPUserAgent
{
    public LocationReply roomLocation;
    public String spi;
    public String language;
    public String scope;
    private UdpClient client;
    private SLPServiceReply slpReply = null;
    public static String SLP_MULTICAST_ADDRESS = "239.255.255.253";
    public static int SLP_PORT = 427;
    Stream fileOut;
    StreamWriter fileWriter;
    DateTime stopCreate;
    /// <summary>
    /// Initializes the UA
    /// </summary>
    /// <param name="roomLocation">the locaiton to search the service for</param>
    /// <param name="spi">the SPI string used to map the public key</param>
    /// <param name="language">the locale</param>
    public SLPUserAgent(LocationReply roomLocation, String spi, String language)
    {
        this.roomLocation = roomLocation;
        this.spi = spi;
        this.language = language;
        this.scope = roomLocation.roomName;
    }
    /// <summary>
    /// Sends a locaiton request and process the reply
    /// </summary>
    /// <param name="request">type of service we are looking for</param>
    /// <returns>array of services available in this location</returns>
    public SLPServiceReply[] findServices(SLPServiceRequest request)
    {

        fileOut = File.Open("\\Temp\\slp_test.txt", FileMode.Append);
        fileWriter.WriteLine("---------");
        DateTime startCreate = DateTime.Now;
        slpReply = null;
        ASCIIEncoding ascii = new ASCIIEncoding();

        byte[] prListByte = ascii.GetBytes(request.prList);
        byte[] serviceTypeByte = ascii.GetBytes(request.serviceType);
        byte[] scopeListByte = ascii.GetBytes(this.scope);
        byte[] predicateByte = ascii.GetBytes(request.predicate);
        byte[] spiByte = ascii.GetBytes(this.spi);
        byte[] langTagByte = ascii.GetBytes(this.language);

/*
   0                   1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |       Service Location header (function = SrvRqst = 1)        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      length of <PRList>       |        <PRList> String        \
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   length of <service-type>    |    <service-type> String      \
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |    length of <scope-list>    |     <scope-list> String       \
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  length of predicate string   |  Service Request <predicate>  \
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  length of <SLP SPI> string   |       <SLP SPI> String        \
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   */
        int srvReqBuffSize =
            2 + prListByte.Length +
            2 + serviceTypeByte.Length +
            2 + scopeListByte.Length +
            2 + predicateByte.Length +
            2 + spiByte.Length;
        byte[] srvReqBuff = new byte[srvReqBuffSize];
        int reqIndex = 0;

        reqIndex = appendBytes(srvReqBuff,
SLPUtil.ToByte16(prListByte.Length), reqIndex);
        reqIndex = appendBytes(srvReqBuff, prListByte, reqIndex);

        reqIndex = appendBytes(srvReqBuff,
SLPUtil.ToByte16(serviceTypeByte.Length), reqIndex);
        reqIndex = appendBytes(srvReqBuff, serviceTypeByte, reqIndex);

        reqIndex = appendBytes(srvReqBuff,
SLPUtil.ToByte16(scopeListByte.Length), reqIndex);
        reqIndex = appendBytes(srvReqBuff, scopeListByte, reqIndex);

        reqIndex = appendBytes(srvReqBuff,
SLPUtil.ToByte16(predicateByte.Length), reqIndex);
        reqIndex = appendBytes(srvReqBuff, predicateByte, reqIndex);

        reqIndex = appendBytes(srvReqBuff, SLPUtil.ToByte16(spiByte.Length),
reqIndex);
        reqIndex = appendBytes(srvReqBuff, spiByte, reqIndex);

        int slpBuffSize = (
              +1                /* Version              */
              + 1               /* Function-ID          */
              + 3               /* Length               */
              + 2               /* Flags                */
              + 3               /* Extension Offset    */
              + 2               /* XID                  */
              + 2 + langTagByte.Length  /* Lang Tag Len/Value   */
              + srvReqBuffSize);       /* Message              */

        byte[] sendBuff = new byte[slpBuffSize];

   /*All requests and services are scoped.  The two exceptions are
    SrvRqsts for "service:directory-agent" and "service:service-agent".*/

        int index = 0;
        byte version = 2;
        index = appendBytes(sendBuff, new byte[] { version }, index);
        byte funID = 1;//SrvRqst=1
        index = appendBytes(sendBuff, new byte[] { funID }, index);
    /*The flags are:  OVERFLOW (0x80) is set when a message's length
    exceeds what can fit into a datagram.  FRESH (0x40) is set on
    every new SrvReg.  REQUEST MCAST (0x20) is set when multicasting
    or broadcasting requests.  Reserved bits MUST be 0.
    */

        index = appendBytes(sendBuff, SLPUtil.ToByte24(slpBuffSize), index);
        int MCAST = 0x2000;
        int flags = 0;
        //request is always multicasted in our case
        flags |= MCAST;

    /*Next Extension Offset is set to 0 unless extensions are used.*/
        index = appendBytes(sendBuff, SLPUtil.ToByte16(flags), index);
        //we don't use extenstions.

        byte extension = 0;
        index = appendBytes(sendBuff, SLPUtil.ToByte24(extension), index);
    /*XID is set to a unique value for each unique request.*/
        int xid = new Random().Next();
        index = appendBytes(sendBuff, SLPUtil.ToByte16(xid), index);

    /*
    Lang Tag Length is the length in bytes of the Language Tag field.
    Language Tag conforms to [7].*/

        //lang tag len
        index = appendBytes(sendBuff, SLPUtil.ToByte16(langTagByte.Length), index);
        //lang tag value
        index = appendBytes(sendBuff, langTagByte, index);
        //add SRVReqst header

        index = appendBytes(sendBuff, srvReqBuff, index);

        //send to port 427

        IPEndPoint ipep = new IPEndPoint(IPAddress.Parse(SLP_MULTICAST_ADDRESS),
            SLPUserAgent.SLP_PORT);
        client = new UdpClient();

        Thread unicastThread = new Thread(unicastReciever);
        unicastThread.Start();

        stopCreate = DateTime.Now;

        TimeSpan span = stopCreate.Subtract(startCreate);
        fileWriter.WriteLine("S1(ms)=" + span.TotalMilliseconds);

        client.Send(sendBuff, sendBuff.Length, ipep);
        System.Console.WriteLine("SLP UA INFO: Service request {0} sent successfully",
            request.serviceType);
        int counter = 1;
        while (counter < 5 && slpReply == null)
        {
            Thread.Sleep(500 * counter);
            System.Console.WriteLine("SLP UA INFO: Waiting for SLPReply.");
            counter++;
        }
        if (slpReply != null)
        {
            slpReply.serviceType = request.serviceType;
            return new SLPServiceReply[] { slpReply };
        }
        else
        {
            return null;
        }
    }

    private static SLPServiceReply processPacket(byte[] buff)
    {
        ASCIIEncoding ascii = new ASCIIEncoding();
        //Extract funciton id
        if (buff[1] == 8)//DA ADV
        {
            //extract URL and save it
            //lang tag header length begins at the 14th byte and extends to the 15th
            byte[] langLenByte = new byte[] { buff[12], buff[13] };
            //int langLen = BitConverter.ToInt16(langLenByte,0);
            int langLen = SLPUtil.FromByte16(langLenByte);
            int header =
                1               /* Version              */
              + 1               /* Function-ID          */
              + 3               /* Length               */
              + 2               /* Flags                */
              + 3               /* Extension Offset    */
              + 2               /* XID                  */
              + 2 + langLen;  /* Lang Tag Len/Value   */
            int urlLenOffset = header +
                2 + /*Error Code*/
                4; /*DA Stateless Boot Timestamp*/

            //URL Length starts at this offset.
            byte[] urlLenByte = new byte[] { buff[urlLenOffset], buff[urlLenOffset + 1] };
            //int urlLen = BitConverter.ToInt16(urlLenByte, 0);
            int urlLen = SLPUtil.FromByte16(urlLenByte);
            //The url string starts at url ofst + url length  (start from 0)
            String url = ascii.GetString(buff, urlLenOffset + 2, urlLen);
            System.Console.WriteLine("SlP UA INFO: Got DA Advert URL: {0}", url);
        }
        else if (buff[1] == 11)//SA ADV
        {
            System.Console.WriteLine("SlP UA INFO: Got SA Advert.");
        }
        else if (buff[1] == 2)
        {
            //SrvReply NOTE: WE assume 1 URL and 1 Auth Block per reply currently!
            //lang tag header length begins at the 13th byte and extends to the 14th
            byte[] langLenByte = new byte[] { buff[12], buff[13] };
            //int langLen = BitConverter.ToInt16(langLenByte,0);
            int langLen = SLPUtil.FromByte16(langLenByte);
            int header =
                1               /* Version              */
              + 1               /* Function-ID          */
              + 3               /* Length               */
              + 2               /* Flags                */
              + 3               /* Extension Offset    */
              + 2               /* XID                  */
              + 2 + langLen;  /* Lang Tag Len/Value   */
            int urlLenOffset = header +
                2 + /*Error Code*/
                2 + /*URLEntity count*/
                1 + /*Reserved*/
                2;  /*Lifetime*/

            //URL Length starts at this offset.
            byte[] urlLenBytes = new byte[] { buff[urlLenOffset], buff[urlLenOffset + 1] };
            //int urlLen = BitConverter.ToInt16(urlLenByte, 0);
            int urlLen = SLPUtil.FromByte16(urlLenBytes);
            //The url string starts at url ofst + url length  (start from 0)
            int urlStarts = urlLenOffset + 2;
            byte[] urlBytes = new byte[urlLen];
            Array.Copy(buff, urlStarts, urlBytes, 0, urlLen);
            String url = ascii.GetString(urlBytes, 0, urlBytes.Length);
            //System.Console.WriteLine("SlP UA INFO: Got SrvRply URL: {0}", url);

            return new SLPServiceReply("", url);

    /*
    * authentication block.
      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Block Structure Descriptor   |  Authentication Block Length  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                           Timestamp                           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |    SLP SPI String Length    |       SLP SPI String        \
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |              Structured Authentication Block ...             \
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                   ASN.1 encoded DSA signature               \
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    */

            int authBlockStarts = urlStarts //the start point of the url
                + urlLen //plus the lenght of the url
                + 1;      //# of URL auths

            int timeStampStarts = authBlockStarts
                + 2 /*BSD*/
                + 2; /*Auth block len*/

            byte[] timeStampBytes = new byte[] { buff[timeStampStarts],
                buff[timeStampStarts + 1], buff[timeStampStarts + 2],
                buff[timeStampStarts + 3] };
            Int32 timeStamp = SLPUtil.FromByte32(timeStampBytes);
            System.Console.WriteLine("Time stamp = " + timeStamp + "\n");
            int spiLenStarts = timeStampStarts
                + 4; /*timestamp*/

            byte[] spiLenBytes = new byte[] { buff[spiLenStarts], buff[spiLenStarts + 1] };
            int spiLen = SLPUtil.FromByte16(spiLenBytes);
            int spiStarts = spiLenStarts
                + 2 /*spi len*/;
            byte[] spiBytes = new byte[spiLen];
            Array.Copy(buff, spiStarts, spiBytes, 0, spiLen);
            String spi = ascii.GetString(spiBytes,0,spiBytes.Length);
            System.Console.WriteLine("SPI= " + spi + "\n");

            int signitureStarts = spiStarts + spiLen;

            System.Console.WriteLine("Auth Block = " +
                BitConverter.ToString(buff, signitureStarts) + "\n");
            int signitureLen = buff.Length - signitureStarts;
            byte[] signitureBytes = new byte[signitureLen];
            Array.Copy(buff, signitureStarts, signitureBytes, 0, signitureLen);
            System.Console.WriteLine("Signiture:" +
                BitConverter.ToString(signitureBytes) + "\n");
            byte[] authDataBytes = new byte[spiLenBytes.Length
                + spiBytes.Length
                + urlLenBytes.Length
                + urlBytes.Length
                + timeStampBytes.Length
                ];
            /*The components listed are
           used as if they were a contiguous single byte aligned buffer in the
           order given.

              URL
                  16-bit Length of SLP SPI String, SLP SPI String.
                  16-bit Length of URL, URL,
                  32-bit Timestamp.
            */
            Array.Copy(spiLenBytes, 0, authDataBytes, 0, spiLenBytes.Length);
            Array.Copy(spiBytes, 0, authDataBytes, spiLenBytes.Length,
spiBytes.Length);
            Array.Copy(urlLenBytes, 0, authDataBytes,
                (spiLenBytes.Length + spiBytes.Length), urlLenBytes.Length);
            Array.Copy(urlBytes, 0, authDataBytes,
                (spiLenBytes.Length + spiBytes.Length + urlLenBytes.Length),
urlBytes.Length);
            Array.Copy(timeStampBytes, 0, authDataBytes,
                (spiLenBytes.Length + spiBytes.Length +
                urlLenBytes.Length + urlBytes.Length),
timeStampBytes.Length);

            System.Console.WriteLine("Auth Data= " +
                BitConverter.ToString(authDataBytes, 0) + "\n");

            //Compute the hash
            SHA1 sha = new SHA1CryptoServiceProvider();
            byte[] digest = sha.ComputeHash(authDataBytes);

            // Load the Public Key
            AsnKeyParser keyParser = new
AsnKeyParser("\\minisip\\public.dsa.cs.ber");

            DSAParameters publicKey = keyParser.ParseDSAPublicKey();
            // Initailize the CSP
            CspParameters csp = new CspParameters();
            //since user profile is not loaded we have to set this flag.
            //csp.Flags = CspProviderFlags.;

            // Cannot use PROV_DSS_DH
            const int PROV_DSS = 3;
            csp.ProviderType = PROV_DSS;

            const int AT_SIGNATURE = 2;
            csp.KeyNumber = AT_SIGNATURE;

            csp.KeyContainerName = "DSA Test (OK to Delete)";

            //
            // Initialize the Provider
            //
            DSACryptoServiceProvider dsa = new DSACryptoServiceProvider(csp);
            dsa.PersistKeyInCsp = false;

            //
            // The moment of truth...
            //
            dsa.ImportParameters(publicKey);

            ///
            //// load the key
            //X509Certificate2 x = new X509Certificate2("ca.pfx");
            //DSACryptoServiceProvider provider =
(DSACryptoServiceProvider)x.PublicKey.Key;

            //convert the signiture from DER to P1363 format
            AsnParser par = new slpua.AsnParser(signitureBytes);
            par.NextSequence();
            byte[] r = par.NextInteger();
            byte[] s = par.NextInteger();
            byte[] b40 = new byte[40];
            Array.Copy(r, 0, b40, 0, 20);
            Array.Copy(s, 0, b40, 20, 20);
            //System.Console.WriteLine("B40: " + BitConverter.ToString(b40));
            DSASignatureDeformatter verifier = new DSASignatureDeformatter(dsa);
            verifier.SetHashAlgorithm("SHA1");
            bool result = verifier.VerifySignature(digest, b40);
            System.Console.WriteLine("****Verify****: " + result);
        }
        else
        {
            System.Console.WriteLine("SLP UA ERROR: Unsupported function id:{0}", buff[1]);
            return;
        }
        return null;
    }
```

I. Partial listing sendAckWithSDP() method

```
void SipTransactionInviteClient::sendAckWithSDP(MRef<SipResponse*> resp, string br){
    MRef<SipCommonConfig *> conf;
    if (dialog){
        conf = dialog->getDialogConfig()->inherited;
    }else{
        conf = sipStack->getStackConfig();
    }
    //Create a request line for ACK similar to "local" <sip:192.168.2.90:5060;lr>
    string requestLine = SipDialog::localResp->getHeaderValueContact()->getString();
    size_t startPos = requestLine.find(":");
    size_t endPos = requestLine.find(";");
    requestLine = requestLine.substr(startPos+1,(endPos-startPos));
    requestLine = requestLine.erase(requestLine.find(">"));
    merr<<"**Request line: "<<requestLine<<end;
    MRef<SipAck*> ack= new SipAck( getBranch(),
        (MRef<SipMessage*>)*SipDialog::localResp,
        requestLine,
        conf->sipIdentity->sipDomain
    );
    //add route headers, if needed
    if( dialog->dialogState.routeSet.size() > 0 ) {
        //merr << "CESC: SipTransInvCli:sendACK : adding header route! " << end;
        MRef<SipHeaderValueRoute *> rset = new SipHeaderValueRoute (dialog->dialogState.routeSet);
        ack->addHeader(new SipHeader(*rset) );
    }
    //ADD SDP WITH CRYPTO ATTRIBUTE:
    // a=crypto:<tag> <crypto-suite> <key-params> [<session-params>]
    //        tag=0,1,2...
    //        crypto-suite= AES_CM_128_HMAC_SHA1_80
    //        key-param = "inline:" <key||salt> ["|" lifetime] ["|" MKI ":" length]
    //example a=crypto:0 AES_CM_128_HMAC_SHA1_80 key-param = inline:d0RmdmcmVCspeEc3QGZiNWpVLFJhQX1cfHAwJSoj|2^20|1:4
    list<MRef<StreamingKey*>>::iterator i;
    string sdpCryptoAttribute = "";
    int j=0;
    for(i=SipDialog::securityContexts.begin();i!=SipDialog::securityContexts.end();i++)
    {
        sdpCryptoAttribute = "a=crypto:";
        sdpCryptoAttribute = sdpCryptoAttribute.append(itoa(j)); //add the tab
        sdpCryptoAttribute = sdpCryptoAttribute.append(" AES_CM_128_HMAC_SHA1_80");//crypto suite
        sdpCryptoAttribute = sdpCryptoAttribute.append(" inline:");//key param
        unsigned char keyMaterial[30];
        int k;
        MRef<StreamingKey *> key = (*i);
        memcpy(keyMaterial,  key->masterKey, 16);
        memcpy(&keyMaterial[16], key->masterSalt, 14);
        //Encode key material in base64
        string encodedKeyMaterial = base64_encode((unsigned char*)keyMaterial,30);
        sdpCryptoAttribute = sdpCryptoAttribute.append(encodedKeyMaterial);//encoded key material
        sdpCryptoAttribute = sdpCryptoAttribute.append("|2^20");//life time
        sdpCryptoAttribute = sdpCryptoAttribute.append("|1:4\n");//MKI not used for now
        j++;
    }
    //append the crypto attribute to the response sdp
    string respSDP = resp->getContent()->getString();
    //remove the a=key-mgmt:mikey
    size_t posKeyMgmt = respSDP.find("a=key-mgmt:mikey");
    size_t posMedia = respSDP.find("m=audio");
    respSDP = respSDP.replace(posKeyMgmt,(posMedia-posKeyMgmt),"");
    if(sdpCryptoAttribute!=""){
        respSDP += "\n" + sdpCryptoAttribute;
    }
    //code to ecrypt and sign SDP using S/MIME.
    //set content lenght
    MRef<SipHeaderValueContentLength*> contLen = new SipHeaderValueContentLength(respSDP.length());
    ack->addHeader(new SipHeader(*contLen));
    //append the result to the ack
    ack->setContent(new SdpContent(respSDP));
    //send method is defined in SipTransaction...
    list<MRef<SipTransaction*>>::iterator k;
    list<MRef<SipTransaction*>> transactions = SipDialog::localDialog->getTransactions();
    for(k = transactions.begin();k!=transactions.end();k++){
        (*k)->send(MRef<SipMessage*>(*ack), true, br);
        wprintf(L"Sending ACK to local %s.\n",(*k)->toaddr->getString().c_str());
        merr<<"***Local ACK Body\n"<<ack->getString()<<end;
    }
}
```

J. OpenSLP configuration files (three files)

```
################################## slp.cnf#################################
#
# OpenSLP configuration file
#
# Format and contents conform to specification in IETF RFC 2614 so the
# comments use the language of the RFC.  In OpenSLP, SLPD operates as an SA
# and a DA.  The SLP UA functionality is encapsulated by SLPIB.
#
###########################################################################
##


#--------------------------------------------------------------------------
--
# Static Scope and Static DA Configuration
#--------------------------------------------------------------------------
--

net.slp.useScopes = Openarea,Desk,Mint,Grimnton,default

#--------------------------------------------------------------------------
--
# DA Specific Configuration
#--------------------------------------------------------------------------
--

net.slp.isDA = false
net.slp.DAHeartBeat = 30

#--------------------------------------------------------------------------
--
# SA Specific Configuration
#--------------------------------------------------------------------------
--

;net.slp.watchRegistrationPID = false

#--------------------------------------------------------------------------
--
# UA Specific Configuration
#--------------------------------------------------------------------------
--

;net.slp.maxResults = 256

#--------------------------------------------------------------------------
--
# Network Configuration Properties
#--------------------------------------------------------------------------
--

;net.slp.isBroadcastOnly = true
;net.slp.passiveDADetection = false
;net.slp.activeDADetection = false
;net.slp.DAActiveDiscoveryInterval = 1
;net.slp.multicastTTL = 255
;net.slp.DADiscoveryMaximumWait = 2000
;net.slp.DADiscoveryTimeouts = 500,750,1000,1500,2000,3000
```

```
;net.slp.multicastMaximumWait = 5000
;net.slp.multicastTimeouts  = 500,750,1000,1500,2000,3000
;net.slp.unicastMaximumWait = 5000
;net.slp.unicastTimeouts  = 500,750,1000,1500,2000,3000
;net.slp.datagramTimeouts = IGNORED
;net.slp.randomWaitBound = 5000
;net.slp.MTU = 1400
;net.slp.interfaces = 1.2.3.4,1.2.3.5,1.2.3.6

#--------------------------------------------------------------------------
--
# Security
#--------------------------------------------------------------------------
--

net.slp.securityEnabled=true
;net.slp.checkSourceAddr=false

#--------------------------------------------------------------------------
--
# Tracing and Logging
#--------------------------------------------------------------------------
--

;net.slp.traceDATraffic = true
;net.slp.traceReg = true
;net.slp.traceDrop = true
net.slp.traceMsg = true


#################################slp.reg###################################
#
# OpenSLP registration file
#
# May be used to register services for legacy applications that do not use
# the SLPAPIs to register for themselves
#
# Format and contents conform to specification in IETF RFC 2614 so the
# comments use the language of the RFC.  In OpenSLP, SLPD operates as an SA
# and a DA.  The SLP UA functionality is encapsulated by the libslp
library.
#
###########################################################################
##

#comment
;comment
#service-url,language-tag,lifetime,[service-type]<newline>
#["scopes="scope-list<newline>]
#[attrid"="val1<newline>]
#[attrid"="val1,val2,val3<newline>]
#<newline>

##Register Speaker Service
service:speaker://speaker1@192.168.2.238,en,65535
scopes=Mint, OpenArea
description=OpenSLP Testing Service
authors=Bemnet

##Register Video Projector Service
service:projector://projector1@192.168.2.238,en,65535
```

```
scopes=OpenArea

##Register Camera Service
service:camera://camera1@192.168.2.238,en,65535
scopes=Mint

##Register Microphone Service
service:mic://mic1@192.168.2.238,en,65535
scopes=Desk
###################################slp.spi#################################
#########
#
# OpenSLP SPI file
#
# Security Parameter Index (SPI) is an unformated string that us used
# by SLP to identify security information used to authenticate SLPv2
# message.  See RFC 2608 for more information.
#
# Format and contents conform of this file are specific to the OpenSLP
# implementation of SLPv2 authentication.  See comments below for more
# explaination of the file format.
#
# NOTE: OpenSLP only supports DSA keys!!!
#
###########################################################################
##
#
#-------------
# File format:
#-------------
# Each line of this file maps an SPI string to a PEM encoded key file.
#
# <PRIVATE|PUBLIC> <spi_string_without_whitespace> <PEM key file>
#
PRIVATE spi1 /home/bemnet/privkey.pem
PUBLIC spi1 /home/bemnet/pubkey.pem
```

K. SER configuration file

```
debug=3
fork=yes
log_stderror=yes

listen=130.237.15.238           # put your server IP address here
listen=192.168.2.238
port=5060
children=4

dns=no
rev_dns=no

#Load general modules
loadmodule "/usr/local/lib/ser/modules/mysql.so"
loadmodule "/usr/local/lib/ser/modules/sl.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/uri_db.so"
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"

#Presence related modules
loadmodule "/usr/local/lib/ser/modules/dialog.so"
loadmodule "/usr/local/lib/ser/modules/pa.so"
loadmodule "/usr/local/lib/ser/modules/presence_b2b.so"
loadmodule "/usr/local/lib/ser/modules/xlog.so"

#Set module-specific parameters
modparam("auth_db|uri_db|usrloc", "db_url",
"mysql://ser:heslo@localhost/ser")
modparam("auth_db", "calculate_ha1", 1)
modparam("auth_db", "password_column", "password")
modparam("usrloc", "db_mode", 2)
modparam("rr", "enable_full_lr", 1)

#presence module related params
modparam("pa", "use_db", 1)
modparam("pa", "db_url", "mysql://ser:heslo@localhost/ser")
modparam("pa", "offline_winfo_timer", 3600)
modparam("pa", "offline_winfo_expiration", 259200)

# mode of PA authorization: none, implicit or xcap
modparam("pa", "auth", "none")
modparam("pa", "winfo_auth", "none")
modparam("pa", "use_callbacks", 0)
modparam("pa", "accept_internal_subscriptions", 0)
modparam("pa", "max_subscription_expiration", 3600)
modparam("pa", "timer_interval", 1)

# module param for b2b
modparam("presence_b2b", "on_error_retry_time", 60)
modparam("presence_b2b", "wait_for_term_notify", 33)
modparam("presence_b2b", "resubscribe_delta", 30)
modparam("presence_b2b", "min_resubscribe_time", 60)
modparam("presence_b2b", "default_expiration", 3600)
```

```
#----Main routing logic--------
route {

      # -----------------------------------------------------------------
      # Sanity Check Section
      # -----------------------------------------------------------------
      if (!mf_process_maxfwd_header("10")) {
            sl_send_reply("483", "Too Many Hops");
            break;
      };

      if (msg:len > max_len) {
            sl_send_reply("513", "Message Overflow");
            break;
      };

      # -----------------------------------------------------------------
      # Record Route Section
      # -----------------------------------------------------------------
      if (method!="REGISTER") {
            record_route();
      };

      # -----------------------------------------------------------------
      # Loose Route Section
      # -----------------------------------------------------------------
      if (loose_route()) {
            route(1);
            break;
      };

      # -----------------------------------------------------------------
      # Call Type Processing Section
      # -----------------------------------------------------------------
      if (uri!=myself) {
            route(1);
            break;
      };

      if (method=="ACK") {
            route(1);
            break;
      } else if (method=="INVITE") {
            route(3);
            break;
      } else      if (method=="REGISTER") {
            route(2);
            break;
      } else if(method =="SUBSCRIBE") {
            route(4);
            break;
      } else if(method =="PUBLISH"){
            route(5);
            break;
      };

      if (uri!=myself) {
            route(1);
            break;
      };
      if (!lookup("location")) {
            sl_send_reply("404", "User Not Found");
            break;
      };

      route(1);
}

route[1] {

      # -----------------------------------------------------------------
      # Default Message Handler
      # -----------------------------------------------------------------
      if (!t_relay()) {
            sl_reply_error();
      };
}

route[2] {

      # -----------------------------------------------------------------
      # REGISTER Message Handler - Currently we implemnet no
authentication.
      # ----------------------------------------------------------------
      sl_send_reply("100", "Trying");

      if (!save("location")) {
            sl_reply_error();
      };
}

route[3] {
      # -----------------------------------------------------------------
      # INVITE Message Handler - Currently we implement no authentication.
      # -----------------------------------------------------------------
      if (uri!=myself) {

            route(1);
            break;
      };

      if (!lookup("location")) {
            sl_send_reply("404", "User Not Found");
            break;
      };

      route(1);
}

route[4] {
      # -----------------------------------------------------------------
      # SUBSCRIBE Message Handler - Currently we implement no
authentication.
      # -----------------------------------------------------------------
      if (!t_newtran()) {
            sl_reply_error();
              break;
      };

      xlog("L_ERR", "PA: handling subscription: %tu from: %fu\n");
      handle_subscription("registrar");
      break;
}

route[5] {
      # -----------------------------------------------------------------
      # PUBLISH Message Handler - Currently we implement no authentication.
      # -----------------------------------------------------------------
      if (!t_newtran()) {
            sl_reply_error();
              break;
      };

      xlog("L_ERR", "PA: handling publish: %tu from: %fu\n");
      handle_publish("registrar");
      break;
}
```

L. Minsip configuration file for the iPAQ PDA

```
<version>
      2
</version>
<network_interface>
      {123456789-1234-1234-1234-123456789ABC}
</network_interface>
<account>
      <account_name>
            ipaq
      </account_name>
      <sip_uri>
            ipaq@192.168.2.238
      </sip_uri>
      <proxy_addr>
            192.168.2.238
      </proxy_addr>
      <register>
            yes
      </register>
      <proxy_port>
            5060
      </proxy_port>
      <proxy_username>
            ipaq
      </proxy_username>
      <proxy_password>
            123456
      </proxy_password>
      <pstn_account>
            no
      </pstn_account>
      <default_account>
            yes
      </default_account>
      <auto_detect_proxy>
            no
      </auto_detect_proxy>
      <register_expires>
            1000
      </register_expires>
      <transport>
            UDP
      </transport>
</account>
<tcp_server>
      yes
</tcp_server>
<tls_server>
      no
</tls_server>
<secured>
      yes
</secured>
<ka_type>
      dh
</ka_type>
<psk>
      No Psk1
</psk>
```

```
<certificate>
\minisip\bob_cert.pem
</certificate>
<private_key>
\minisip\bob_key.pem
</private_key>
<ca_certificate>
\minisip\ca_cert.pem
</ca_certificate>
<dh_enabled>
      yes
</dh_enabled>
<psk_enabled>
      no
</psk_enabled>
<check_cert>
      no
</check_cert>
<local_udp_port>
      5060
</local_udp_port>
<local_tcp_port>
      5060
</local_tcp_port>
<local_tls_port>
      5061
</local_tls_port>
<sound_device>
      wave:test
</sound_device>
<mixer_type>
      simple
</mixer_type>
<codec>
      G.711
</codec>
<phonebook>
      file:\\./minisip.addr
</phonebook>
<auto_answer>
      no
</auto_answer>
<use_srtp>
      yes
</use_srtp>
<use_ipsec>
      no
</use_ipsec>
<use_stun>
      no
</use_stun>
<stun_server_autodetect>
      no
</stun_server_autodetect>
<stun_server_domain>
</stun_server_domain>
<stun_manual_server>
</stun_manual_server>
<ringtone>
</ringtone>
```

TRITA-ICT-EX-2009:63

www.kth.se