# A Meeting Detector to Provide Context to a SIP Proxy

XUELIANG REN

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2008

COS/CCS 2008-24

# A Meeting Detector to Provide Context to a SIP Proxy

Xueliang Ren

Oct. 25, 2008

Supervisor & Examiner

Professor Gerald Q. Maguire Jr.

Submitted in partial fulfillment of

the requirements for the degree of

Master of Science (Information Technology)

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

# Abstract

As sensing technology develops, it plays an important role in context-aware systems. Using context information improves the user experience of ubiquitous computing. One use of sensed information is to detect a meeting in progress in an office or a conference room. In our system, sensors gather context information from an office environment and act as a presence user agent to update a presence server with context changes. These context changes can be utilized by context-aware services. The presence messaging uses the SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) protocol and the presence information is described in eXtensible Makeup Language (XML) format.

In this thesis we present a context-sensing component that recognizes meetings in a typical office environment. A context-aware system is able to use this occupancy information to infer that the room is empty, an individual is alone in the room, or a meeting is taking place in the meeting room. Context-aware services might utilize this environmental information to automatically forward a user's incoming calls to their voice mail server. This and other example applications were developed to show the usefulness of this context information.

## Sammanfattning

Så som sensor tekniken utvecklas, spelar de en viktig roll i kontextmedvetna system. Genom att använda kontextuell information förbättras användarupplevelsen av 'ubiquitous computing'. Ett användningsområde för sensorinsamlad information är att upptäcka ett möte som pågår i ett kontor eller konferenslokal. I vårt system samlar sensorer information från en kontorsmiljö och uppdaterar en närvaroserver med kontextuella förändringar. Dessa förändringar kan sedan utnyttjas av kontextmedvetna tjänster. För att förmedla den närvarostatusen använder närvaroservern SIP och ’Presence Leveraging Extensions’ (SIMPLE) protokoll. Närvaro information levereras i 'eXtensible Makeup Language' (XML) format.

I denna avhandling presenterar vi en kontextsensorkomponent som känner av möten i en typisk kontorsmiljö. Ett kontextmedvetet system kan använda denna komponent för att dra slutsatsen att lokalen är tom, en person är ensam i lokalen, eller ett möte äger rum i lokalen. Kontextmedvetna tjänster kan utnyttja denna information för att automatiskt vidarebefordra en användares inkommande samtal till deras röstbrevlåda. Detta och andra exempel, har utvecklats för att visa nyttan av denna kontextuella information.

## Acknowledgments

I would like to express sincere gratitude to my supervisor and examiner, Professor Gerald Q. "Chip" Maguire Jr. with his kind help and encouragement to help me complete this thesis project. The valuable comments and suggestions from his feedback did a great help for my report. All of his academic achievements, educational philosophy, and gentle attitude impressed me a lot. I really had a rewarding experience at Wireless@KTH with the kind guidance from him.

My thanks also come to all my friends and colleagues who supported me through this period at school.

Finally, and as always, I would like to thank my beloved parents for their endless support and encouragement during my life.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# 1. Introduction

## 1.1 Problem Statement

Nowadays, modern sensing technologies are more and more connected with social demands and commercial use. It is increasingly viable to sense context in a variety of environments. This can be used to enable context-aware services, which are highly desirable as people wish to find new ways to make life easier. In fact, they expect that life should become easier and easier with improved technology.

The need for context is even greater when we move into non-traditional, off-the-desktop computing environments. From our own experiences on the campus, it is easy to see many potential uses for context. Whether to organize a group meeting or to find a place for a couple hours of academic study on the campus, unoccupied meeting & project rooms are always in demand. Similar needs can also occur in corporate buildings or even in hotels. The difficulties in trying to find an available study area or meeting room on campus initially motivated our work to design a system that could facilitate the search, saving both time and aggravation. While it might seem that a simple scheduling system would be sufficient for our needs, we observed that quite frequently rooms where reserved but not utilized for the entire period for which they were reserved. To be able to detect that a room might be reserved, but not currently utilized suggested that sensors be used to determine room occupancy. This leads to the realization that room or area status information could be used in other context-aware services. Therefore the idea of connecting a meeting detector with a room scheduling system came about naturally. This is not a new thought, it was a goal of the earlier Adaptive and Context Aware Systems project (ACAS) at KTH; but the necessary technology did not exist. This thesis project is another step towards establishing the required technology.

## 1.2 Objectives

In this thesis, we are interested in detecting a meeting (i.e., detecting that a room is in use for a meeting) in an office environment. We expect this to be useful for two classes of applications: (1) applications that help the user to control devices in the room, such as projector, lighting, heating, cooling, and fresh air circulation; as well as programming the user's phone (or telephone proxy) to send incoming calls to voicemail; and (2) applications that help a central room scheduling system to monitor the occupancy status of every room in order to facilitate users making bookings via a room reservation service. A third class of applications concerning management of rooms (in terms of long term planning, security, and safety) are outside of this thesis project; but could be topics of future projects.

The main objective of our work is to develop, evaluate, and improve meeting detector based occupancy sensing in conjunction with a room scheduling context-aware system.

Many context-aware systems have been developed in the research community using various sensors and acting on different sources of context information. In this masters thesis, we wish to implement a meeting detector system at the KTH Center for Wireless Systems (Wireless@KTH). The whole system should consist of independent nodes (wireless sensors), a presence server, and some example applications.

The room occupancy information is gathered by the sensor nodes and sent to a central receiver (the presence server). Daniel Hübinette developed a prototype of the occupancy sensor system in his masters thesis project in 2007 [1]. In parallel with this, detailed research on the presence server was done by Mohammad Z. Eslami and described in his thesis "A Presence server for Context-aware applications" [2]. There are also some applications implemented by earlier students, with more being implemented today by other students. In this context-aware system, the room status information may be processed and displayed directly on a user's display, as suggested in the thesis by Yu Sun [3]. In addition to providing occupancy data, a room reservation service can be developed to make it convenient to (re-)book rooms which are reserved but not in use; leading to a smart meeting room system.

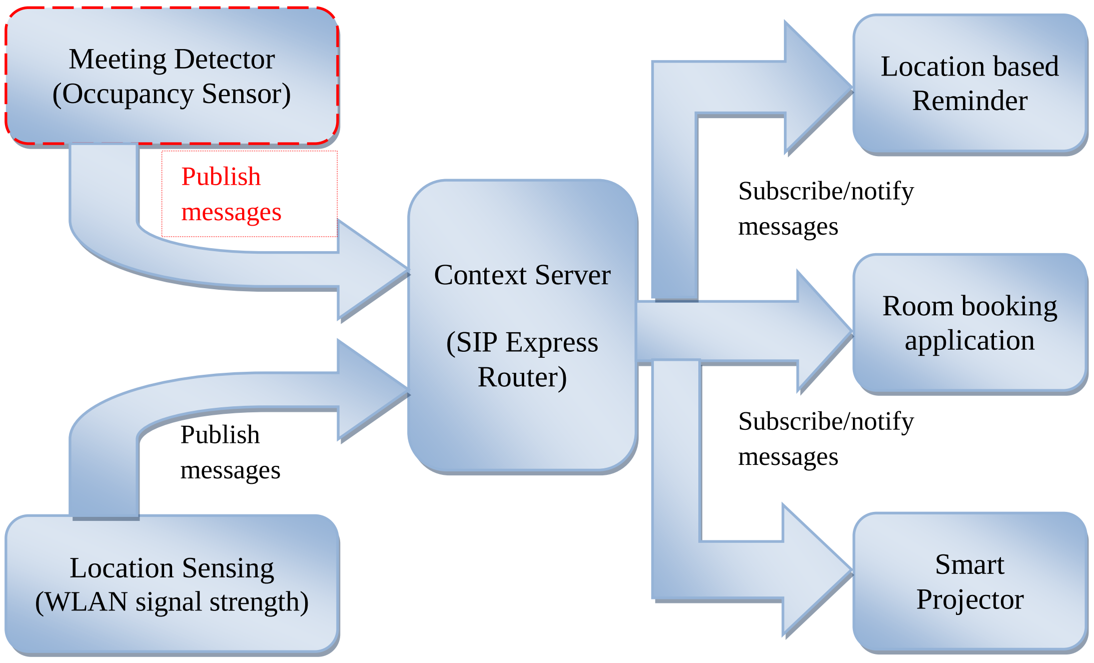


Figure 1.1 Context-aware System Overview

Figure 1.1 shows the complete architecture of our context-aware system. As we can see, the Meeting Detector and Location Sensing subsystem publishes messages as inputs to the Context Server, which can send notify messages to applications such as a Location-based Reminder or Smart Projector. The red highlighted portion of the figure is our specific part of the overall system.

## 1.3 Organization of this Thesis

In this thesis, we will present some background information about context-aware systems, introduce some related technologies, and describe some existing solutions for room occupancy and meeting status detection in Chapter 2. Following this, in Chapter 3, we will integrate the meeting detection systems together with a context-aware infrastructure in order to publish room occupancy and meeting status information to context-aware services and applications. The testing of the system and analyses of experimental results will be presented in Chapter 4, as well as an evaluation and suggested improvements in this meeting detector system. The thesis will conclude with a summary of our conclusions and suggested future work in Chapter 5.

# Chapter 2

# 2. Background and Related Work

In this chapter, we will introduce background information about context-aware systems, including the basic context-awareness architecture and examples of sensor systems. Related technologies and research will also be presented at the end of this chapter.

Context and context-awareness have been central issues in ubiquitous computing research for the last decade. In order to get a better understanding of a context-aware system, some concepts that are commonly used in this area have to be explained first.

## 2.1 Context and Context Awareness

### 2.1.1 What is a Context?

With computers being used in such a wide variety of situations, interesting new problems arise and the need for context is clear: users are trying to obtain different information from the same services in different situations. Context can be used to help determine what information or services to make available or to bring to the forefront for users.

Most researchers have a general idea about what context is and use that general idea to guide their use of it. However, a vague notion of context is not sufficient; in order to use context effectively, we must attain a better understanding of what context is.

From a user's context, we potentially get information such as identity, location, time, temperature, and so on. The use of context is becoming increasingly important in the fields of handheld and ubiquitous computing, where the user's context may change rapidly.

In [4], Schilit and Adams introduce three important aspects of context: where the user is, who the user is with, and what resources are nearby. They define context to be the constantly changing execution environment. They include the following pieces of the environment:

a) Computing environment: available processors, devices accessible for user input and display, network capacity, connectivity, and costs of computing

b) User environment: location, collection of nearby people, and social situation

c) Physical environment: lighting and noise level

According to Dey and Abowd's discuss, we have a more clearly understanding, they define the word "context" as follows:

*"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. "* [5]

### 2.1.2 Definition of Context Awareness

Context-awareness is a kind of intelligent computing behavior. The term context-awareness was introduced by Schilit in 1994 to describe a new class of computer software application that exploits the changing environment of a mobile computer user.

Anind K. Dey defines context aware in his doctor thesis "Providing Architectural Support for Building Context-Aware Applications" [6]

*"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."*

In my understanding, for the computing systems, context-awareness is the capability to provide relevant services and information to the users based on their situational conditions (i.e., contexts). For example, context-awareness enables a person to follow an ongoing conversation, and context awareness can help to guide the appropriate behavior of a student when the student enters a building, a classroom or passes by the library.

In [5], Gregory D. Abowd and Anind K. Dey mentions three important context awareness behaviors are the presentation of information and services to a user, automatic execution of a service, and tagging of context to information for later retrieval.

The context-aware service in sensor based ubiquitous network consists of a sensor platform, a context-aware framework and an intelligent agent. The sensor platform collects raw data for context-aware services, the intelligent platform aware residents and building context, then triggers intelligent and automatic services according to the situations related to user, building environments and so on.

## 2.2 Context-aware System

The convergence of cellular telephony, pocket PCs, location information, and other sensor data have well provided a basis for context-aware solutions. Many different architectures and middleware systems have been developed to support context-aware systems over recent years. Let us examine some applications and scenarios in smart environments before including context-aware system architectures.

### 2.2.1 Context-aware System Applications

Researchers believe that the use of context can help computing systems to anticipate our needs and act on our behalf. The growing demand for untethered access within home, office, and outdoors has provided boundless opportunities for creating new context-aware systems and services. Location and user identity are commonly used in context aware systems - such as smart space applications.

In the project Teleporting - Making Applications Mobile [7] at the Olivetti Research Laboratory (ORL), a call forwarding system uses a person's location information to decide where the incoming calls to the person should be routed to. They use an automatically maintained a database of the location of equipment and people within the building for this context-aware teleporting system.

Asthana et al. describe a shopping assistant system in [8]. In this system, a hand-held wireless communications device, the Personal Shopping Assistant (PSA), is used to communicate with the Personal Shopping Server which tracks the shoppers' identities. The centralized server maintains a customer database and uses each shopper's identity information to provide services, such as recommending products and helping the customers to locate shelved items.

### 2.2.2 Scenarios of Context-aware Systems in a Smart Meeting Room Environment

Consider some typical scenarios of a context system in a smart meeting room (based upon the thesis project of Lidan Hu [9]) [10].

a) In the smart context-aware environment, as a speaker enters the room, the intelligent sensor in the room detects and recognizes her presence and reasons about her intention. Knowing she is the speaker, the room concludes that her intention is to give a presentation. The system may greet her by saying "Welcome" on the display screen in the room. Based on the profile information of this person, the room informs the projector device of the URL from which the presentation slides can be fetched (the slides are uploaded to a server before). After the slides have been downloaded, the projector device sets up the presentation automatically for the speaker and presents the first slide at the scheduled time.

b) Knowing that her slides are ready, the speaker could press a button on her cellular phone or PDA to signal the room that she wishes to start her presentation. She does not even need to use her laptop during the process. At this time, the room may dim the lighting (if it senses that the lighting condition is too bright for the audience to easily view the projected presentation).

c) During the presentation, the smart room detects the absence of some people who previously expressed their interest in the presentation. Knowing that they had planned to attend, the room utilizes their contact information to inform a meeting minutes agent to send copies of the recorded presentation and the meeting minutes to those people. Note that alternatively the users could receive reminders that the meeting has started (via SMS, e-mail, synthesized phone call, etc.) - this reminder might even include the SIP URL to be able to remotely join the presentation (as a multimedia session).

d) As the presentation comes to an end, the speaker leaves the conference room, but left behind her PDA in the room. The room detects the PDA's presence. Knowing the device is not co-located with its owner, the room sends a notification to the speaker via text messaging. The user's SIP proxy forwards this text message to the user's cellular phone, her desktop SIP user agent, etc. The system can also automatically return the lights to the appropriate level for the rooms next use (if any), instruct the presentation system to delete the speakers slides (if they are not to be retained), terminate the multimedia session if there were any participants, ... .

This is a typical smart meeting environment. It assists users by managing the scheduling of the meeting, assisting the speakers, and can provide the attendees with an up to date agenda and updated documents related to the meeting. It can even help take on some of the responsibilities of the moderator in a standard meeting, freeing the human moderator to handle the unexpected.

### 2.2.3 Context-aware System Architectures Based on Acquisition Methods

In a context aware system, there are several processes that should be considered. Firstly, the system needs to acquire context data via different type of sensors. Secondly, the context data should be stored and processed for later use. Finally, the system must distribute the context data to applications that need the information.

Typically context-aware systems acquire contextual information via sensors. We can categorize sensors based on different architectures. Usually these sensors are hardware sensors (e.g. thermal sensors, temperature sensors, touch sensors and audio/video sensors). Context sensors may also be software programs that aggregate information acquired from the hardware sensors and instrumented software to form more specialized knowledge. Different context-aware systems have explored different architectures and methods to acquire context. Normally, the context-aware framework perceives context based upon raw context information from a sensor platform. An intelligent agent utilizes this data to make inferences, in order to trigger automatic services based upon the context information from the context-aware framework.

In [10], Harry Chen defines the following three categories of context acquisition methods: (a) Direct access to sensor, (b) Middleware infrastructure, and (c) Acquire context from a context server. In the context aware Pocket PC designed by Hinckley and Jeff Pierce [11], the context-aware system acquires the states and the position of the device by directly accessing the on-board sensors – proximity range sensors, touch sensors, and tilt sensors. Anind K. Dey introduced Context Toolkit [12] in his PhD research, which is a middleware infrastructure for supporting context acquisition.

In this meeting detector system, we decide to use the middleware infrastructure. The idea is that middleware infrastructures facilitate sensing. This approach introduces a layered architecture, which supports better scalability, extensibility, and reusability. In this infrastructure, context aware applications' implementations can focus on how to use context, rather than how to acquire context.

As we can see in Figure 2.1, there are three levels: sensors, middleware, and applications. The middleware includes one or more context collectors, processors, repositories, and distributors. The sensors collect context information from environment. Then the context information is delivered to higher layers and stored in a context repository. At the same time, the context information is processed by the context processor for later distribution. Finally, the context distributor selects and distributes the context information to applications and their users.

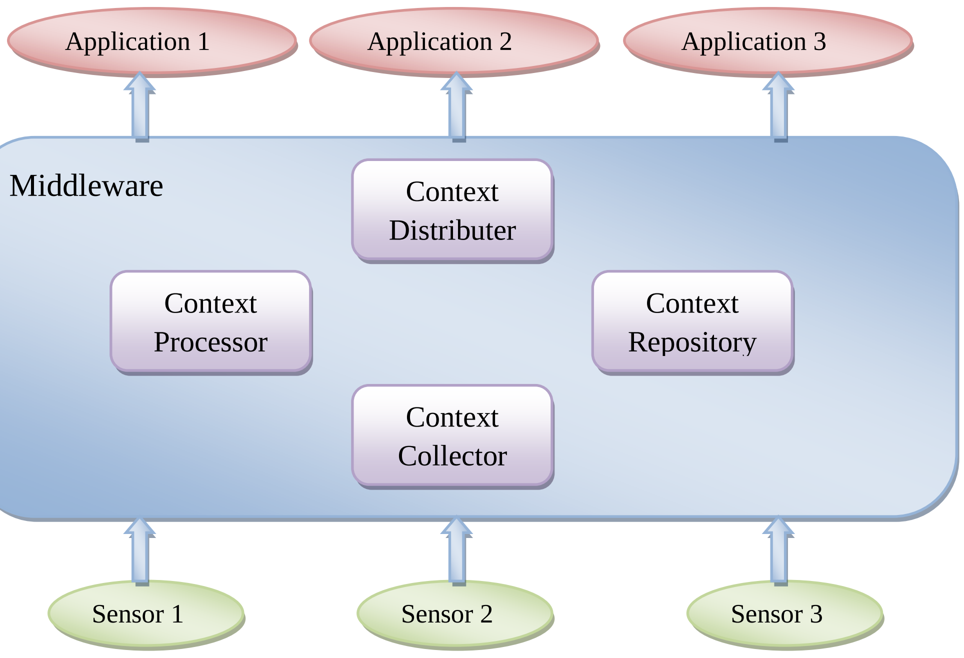


Figure 2.1 Middleware Architecture [2]

In this acquisition approach it trades computation resources for development convenience. In order to maintain a generic programming interface between the high-level applications and the low-level sensors, a certain amount of computation & communication resources (e.g., CPU power, memory, network bandwidths) must be allocated for the middle-ware's operations. This may lead to problems when there are some insufficient resources, for example for devices that have limited resources such as PDAs and cellular phones.

Many modern context aware systems use a middleware infrastructure because the presence server allows multiple applications to reuse the context data that can be distributed. This infrastructure was used in Mohammad's thesis project. He also explained the details of the other schemes in his thesis paper [2]. The communication between the middleware infrastructure and the sensor system (which collects context information) is one of main focuses of our work.

## 2.3 Sensor System

A sensor system is used to gather the necessary information about a user's environment from specific sensors. Information gathering, data corrections and management are the main objectives of the sensor networking.

### 2.3.1 Sensors

In a pervasive computing environment, sensors are often used to detect the presence of people in a building. For example, Radio Frequency Identification (RFID) sensors can detect the presence of RFID Tags and infer the presence of people who are expected to be wearing them, and Bluetooth sensors can detect the proximity (presence) of Bluetooth-enabled personal devices and infer the presence of the device owners. More details about location sensing system can be found in [13].

In order to detect the occupancy in an area, two approaches can be used. The first approach is to detect, locate, and track persons or movements within a zone. The other approach is to record the event when someone passes by a fixed point as they enter or exit the area.

Context information can be collected from variety of sources. In [14] Baldauf and Dustdar define three different types of information collecting sensors can do. These are physical, virtual, and logical sensors. For different purposes, one or more different physical and virtual sensors can be used to gather data for the context-aware systems. The sensed information is classified as the base information. The sensor information manager is in charge of collecting the raw data and managing the collected information.

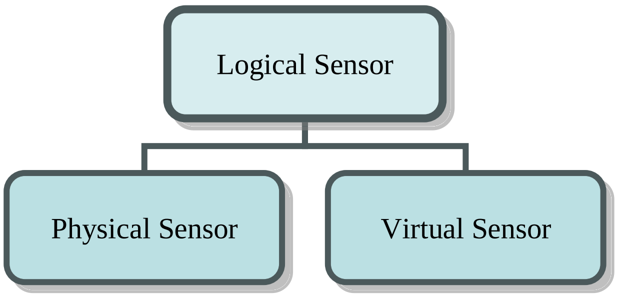


Figure 2.2 Sensor Entities

A generic process for determining occupancy in a room can be divided into the following phases: data gathering, data analysis, and data distribution as shown in Figure 2.3. The data gathering phase is where data is collected and prepared for data analysis. The data analysis phase is where occupancy is determined with the aid of the collected data. The data distribution phase is where the result of the analysis is made available for use in a context-aware system. The three phases have to occur in the order specified.

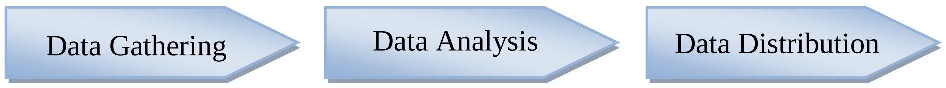
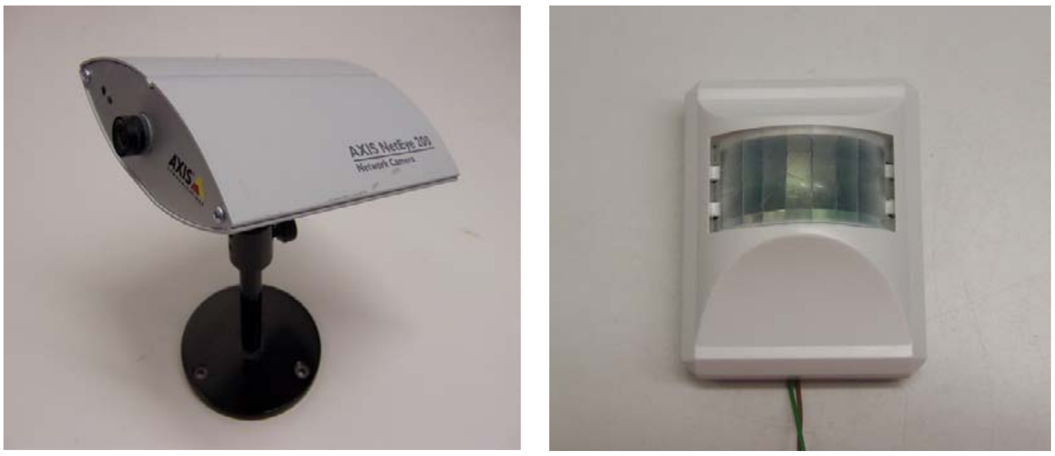


Figure 2.3 Generic Sensor System as a Process

In a room occupancy system, a physical sensor may be used to detect the occupancy of the room, such as a camera or thermal detector. Figure 2.4 (a) shows an AXIS NetEye digital camera [15]. It features a built-in web server. With this network camera, users (and applications) can take and view pictures remotely over the network with a standard web browser. The data collected by the camera can be analyzed and used as context information.



(a) NetEye Digital Camera (b) Infrared Intrusion Detector

Figure 2.4 Sensor Detector

In Daniel Hübinette's thesis [1], he described three types of sensors. The advantages and disadvantages off each of the technologies were also explained. After the comparisons and analyses, he chose thermal detectors for his occupancy sensor system. He focused on designing and evaluating an occupancy sensor system prototype. The prototype was designed to detect, determine, and distribute room occupancy context data. In this case, the context data refers to the occupancy values: zero, one, or many entities in the room. For the detection hardware, a Velleman Passive Infrared (PIR) Mini Intrusion Indoor Detector, HAA52, was selected. It employs a dual element pyroelectric sensor that allows it to be attached to a surface or corner, between one to five meters above floor level. A picture of this detector is showed as Figure 2.4 (b). Details of this sensor are provided in his thesis.

### 2.3.2 Sensor System Architecture

In [14], Baldauf and Dustdar propose a layered conceptual architecture which separates detecting and using context to improve extensibility and reusability of the system. The first layer consists of a collection of different sensors. It is notable that the word "sensor" not only refers to sensing hardware, but also to every data source which may provide usable context information. The context information may be collected by various types of sensors. The second layer, Raw data retrieval layer, is responsible for the retrieval of raw context data. After receiving the data, Raw data retrieval layer provides a sensor application programming interface abstraction for the data preprocessing. The third layer, Preprocessing layer, is responsible for reasoning and interpreting of the context data from the lower layers. The fourth layer, Storage and Management layer, organizes the gathered data and offers them via a public interface to the client. In this layer the context information is stored and managed for later use. The actual reaction to different events and context states is implemented in a fifth layer, Application layer. Context-aware applications can be developed based on the collected and processed data.

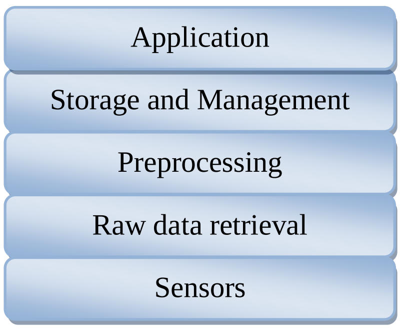


Figure 2.5 Sensor System Layered Conceptual Architecture

### 2.3.3 Occupancy Sensor as a Presence User Agent

The occupancy information should be sent to entities such as the context server after the data have been gathered and analyzed. As we can see in Figure 2.6, the context server works as a Presence Agent (PA) in a context-aware system. At the same time, the occupancy sensor works as a Presence User Agent (PUA) which is responsible for transferring the context information to the context server (or PA). The presence information exchange mechanism and related technologies will be explained in the following section.

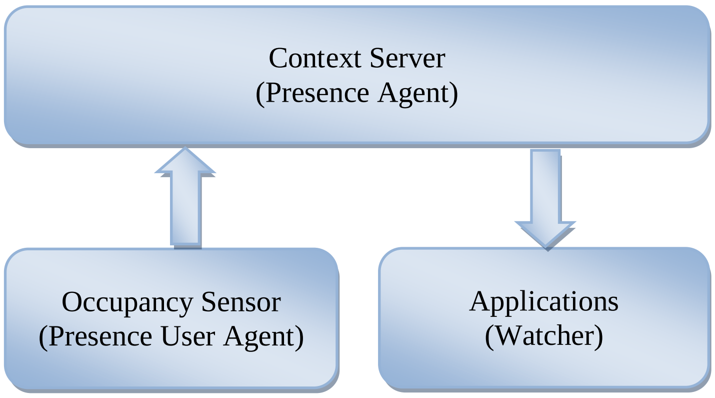


Figure 2.6 Presence User Agent

## 2.4 Related Technologies

In context-aware systems, the presence information needs to be detected, described, stored, and exchanged in a way that can be understood and processed by different types of systems. Some of the commonly used technologies in this process are described in the subsections below.

### 2.4.1 XML

XML [16], short for eXtensible Markup Language, is a generic markup language for data representation. It is a simple, very flexible text format. It started as a simplified subset of the Standard Generalized Markup Language (SGML) [17], and is designed to be human-readable and meet the needs of large-scale electronic publishing. XML became a W3C Recommendation in February 1998 [18].

XML is designed to make parsing easy to implement. It is a markup language unlike HTML (which is interpreted by a browser to display data from a webpage - as HTML contains not only the content but also describes the web page layout). XML is designed to structure, store, transport, and encode information. A variety of application languages can be implemented in XML by adding semantic constraints, these include: XHTML, RSS, MathML, GraphML, Scalable Vector Graphics, MusicXML, and thousands of others [16].

XML is a markup language for documents containing structured information. It plays an important role in the exchange of a variety of data on the Internet. Today, XML is the most common data exchange format and is used in many aspects of web development.

An extension of XML is used in the Markup Scheme Model [19] which is commonly used for context modeling. This allows new context information to be easily utilized. To facilitate the future development of context-aware applications, users should define extensible tags in an XML Document Type Definition (DTD) file to describe the context information. Having a formal description in a DTD file enables programs to check the XML grammatical correctness of a document using this DTD.

### 2.4.2 SIP

#### *2.4.2.1 What is SIP*

The Session Initiation Protocol (SIP) [20] is a signaling protocol for Internet conferencing, telephony, presence, event notification, and instant messaging. The Session Initiation Protocol (SIP) working group is chartered to maintain and continue the development of SIP, currently specified as a proposed standard in RFC 3261 [21], and its family of extensions. SIP can establish sessions for video conference, interactive gaming, and call forwarding over IP networks. While SIP was desired to be used for multimedia call session setup and control over IP networks. Today, SIP is also widely used (for many different types of applications) since it is very easy to find a SIP stack to develop the applications desired - while leveraging the power features of SIP (such as user and device mobility) and leveraging the increasingly common SIP infrastructure.

In [22], Ubiquity Software Corporation presents an overview of SIP. A detailed and technically informed introduction to SIP ecosystem is described by Sinnreich and Johnston in [23]. A list of SIP's major features are:

a) SIP is a text-based protocol for initiating interactive communication sessions between end users. This makes SIP both flexible and readily extensible. SIP is the first protocol to enable multi-user sessions regardless of media content.

b) SIP is designed to be independent of the lower-layer transport protocol, which allows it to take advantage of new transport protocols.

c) SIP is a request-response protocol that closely resembles two other Internet protocols, the web's Hyper Text Transfer Protocol (HTTP) formatting protocol and the Simple Mail Transfer Protocol (SMTP) email protocol; consequently, SIP fits comfortably alongside other Internet applications and leverage user familiarity with URLs, email addresses, client-server protocols, etc. Using SIP, telephony becomes another network application and can be easily integrated with other Internet services.

d) SIP is flexible, extensible, and open, and it is galvanizing the power of the Internet and fixed and mobile IP networks to create a new generation of services.

e) SIP is analogous to HTTP in the way it constructs messages, so developers can easily and quickly create applications using popular programming languages such as Java.

Based upon these features, SIP readily supports: (a) Notion of presence and user location mechanisms; (b) Application-layer routing (including forking) and message processing (e.g., CPL - see section 2.4.6). These two characteristics make SIP very suitable for our context-aware system - as we do not have to worry about user or device mobility (as SIP takes care of this for us) - and we can easily extend CPL to make decisions about forwarding calls based upon occupancy context information.

#### *2.4.2.2 SIP Architecture*

SIP is a signaling and control protocol for multimedia sessions. The SIP architecture provides personal, terminal, and session mobility with a readily available infrastructure.

From an architecture standpoint, the basic components of a SIP network can be grouped into two categories: the SIP user agents and the SIP network servers. The user agent is the end system component for the session (such as a voice over IP (VoIP) call) and the SIP server is the network device that handles the signaling associated with multiple sessions. The user agent itself has a client element, the User Agent Client (UAC) and a server element, the User Agent Server (UAS). The UAC initiates requests, and the UAS generates responses to received requests. This allows peer-to-peer sessions to be created using a client-server protocol. Figure 2.7 shows the basic architecture and operations of SIP. We explain the components of this architecture following this figure.

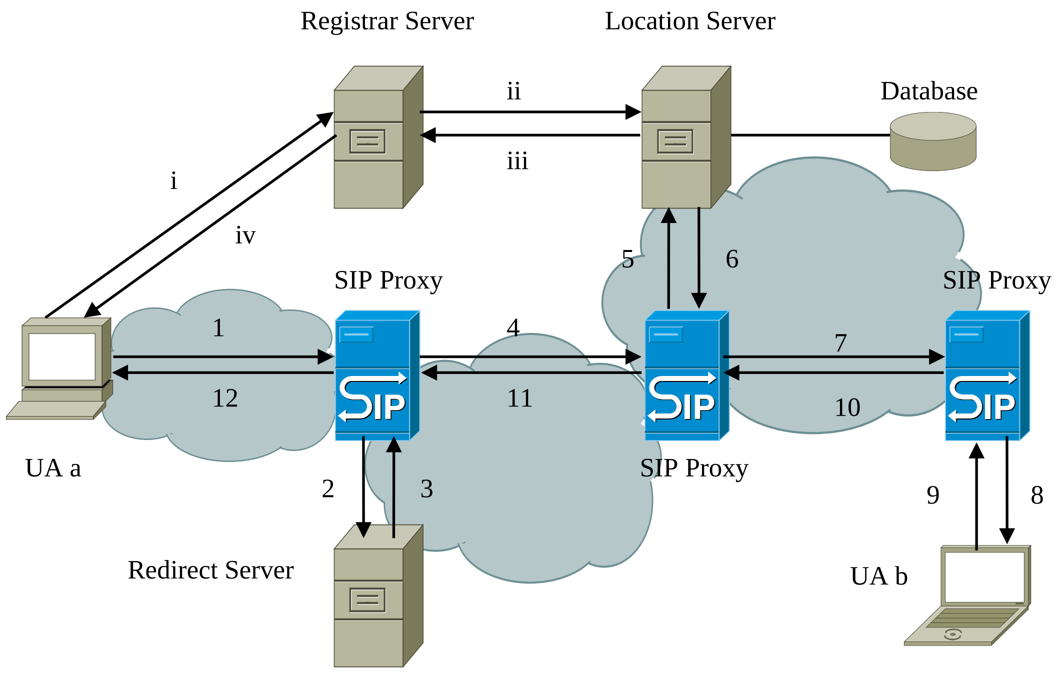


Figure 2.7 SIP Architecture and Operations (Adapted from [24])

| | |
|---|---|
| **User Agent Client** | A logical function that creates a request, then as a client sends this request |
| **User Agent Server** | The logical function that generates a response to a SIP request |
| **SIP Proxy** | An intermediary that forwards or proxies the request from a UA or proxy to another location |
| **Redirect Server** | A server that redirects a request to a user agent for direct routing to complete this request |
| **Registrar Server** | A server that receives SIP registration requests and updates the UA's information into a location server or other database |

In [24], ChenXin Zhang introduces other advanced servers, including:

| | |
|---|---|
| **Presence Server** | Exploits the users' presence information to facilitate any type of Internet or telecommunication application |
| **Location server** | A SIP/SIMPLE server to maintain location information for currently registered SIP user agents |

The main function of the SIP servers is to provide name resolution; maintain knowledge of the user agent's location, since the caller is unlikely to know the current IP address or host name of the called party; and to pass on SIP messages to other servers using next hop routing protocols. The steps from i to iv in Figure 2.7 show how a SIP entity registers it location with a SIP Registrar.. The detailed process of SIP signaling to set up a session between two users is illustrated in steps 1-12 in Figure 2.7; in this case the SIP messages traverse from $UA_a$ to $UA_b$ via several SIP proxy servers.

### *2.4.2.3 SIP Messages and Process*

SIP communication occurs through two types of messages: requests or responses. The UAC makes requests and the UAS returns responses to client requests. The six main types (methods) of requests are [25]:

| | |
|---|---|
| INVITE | Indicates a user or service is being invited to participate in a session |
| ACK | Confirms that the client has received a final response to an INVITE request |
| BYE | Terminates a session and can be sent by either the caller or the callee |
| CANCEL | Cancels any pending searches but does not terminate a session that has already been accepted |
| OPTIONS | Queries for the options and capabilities of a server |
| REGISTER | Registers the UA with the SIP registration, which updates its location in the location server |

SIP uses requests and responses to establish communication between two or more end points. An invitation to a session occurs when one SIP end point (user A) invites another SIP endpoint (user B) to participate in a session. During this process, user A sends an INVITE message requesting that user B joins a particular conference or to establish a two-party conversation. If user B wants to join the session, it sends an affirmative response. Otherwise, it sends a failure response. Upon receiving the response, user A acknowledges the response with an ACK message. If user A no longer wants to participate in the session, it sends a BYE message instead of an ACK message.

### 2.4.3 SIP Express Router

SIP Express Router (SER) is a SIP proxy (router) which was developed by iptel [26] based upon the SIP standard. SER is an open source SIP server which is free and highly configurable, as well as high-performance. SER is an extremely scalable and flexible SIP server. SER is complete and can support SIP according to RFC 3261 over TCP and UDP. In [27], Jiri Kuthan introduces SIP and SER.

The most important configuration file of SER is "ser.cfg". It may be thought of as the brains of the SIP router (due to its fourth section). This file is divided into four parts: Global Configuration, Module Loading, Module-specific Parameter, and Routing Logic. The second and third parts of the "ser.cfg" configuration file control which modules should be loaded and defines how these modules should behave by setting module variables. SER is being used by many SIP device vendors. Using SER helps us to achieve excellent interoperability.

Mohammad Z. Eslami [2] both introduced SER into our context-aware system and showed that SER had sufficient performance for use in this system. Therefore, we will continue to use SER to register users in a database (which acts both as a general database and as the SIP location server) enabling SIP messages to be routed between clients, service agents, applications, and sensors.

### 2.4.4 SIMPLE

The IETF has produced many specifications related to Presence and Instant Messaging with the Session Initiation Protocol. Collectively, these specifications are known as SIMPLE, which stands for Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions [29]. The protocol suite is based on SIP, so it enables messages to be exchanged within a SIP session and provides a subscription based framework for an event notification. SIMPLE is an extension of SIP to deliver both instant messaging (IM) and presence information. Presence information is much broader than just IM, and it enables communications using voice and video along with IM. In [29], M. Day et al. define a Presence Service which accepts presence information, stores it, and distributes it.

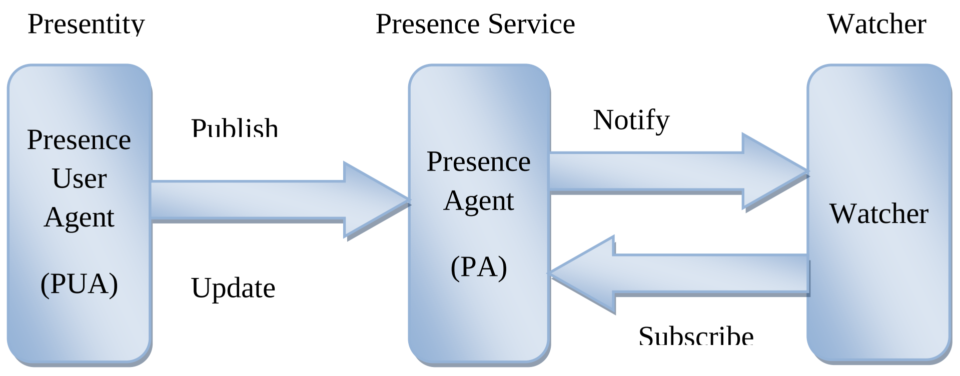


Figure 2.8 Presence and Instant Messaging Model

Figure 2.8 shows how the presence messages are exchanged according to the SIMPLE standard. The Presence Service (or Presence Agent (PA)) has two distinct sets of "clients". One set of clients, called a Presentity or Presence User Agent (PUA), *provides* presence information to be stored and distributed. The PUA detects the context information and updates the PA via a PUBLISH message. The other set, called Watchers, *receives* presence information from the service. Two kinds of watchers are introduced in this model: fetcher and subscriber. The fetcher simply requests the presence information from the Presence Agent. In contrast, the subscriber is interested in updates to presence information, so it sends a subscribe message in order to receive updates from the Presence Agent.

### 2.4.5 PIDF

In [30], Hiroyasu Sugano et al. define the Presence Information Data Format (PIDF) as a common presence data format for Common Profile for Presence (CPP) [31] -compliant presence protocols. It provides a means for presence information to be transferred across CPP-compliant protocol boundaries without modification. In the model of Presence and Instant Messaging shown in Figure 2.8, the presence information is sent by a PUA. This information is in the form of an XML presence document utilizing the Presence Information Data Format (PIDF).

PIDF encodes presence information in XML. XML is considered as the most desirable encoding because it has a hierarchical structure and can be readily extended. A presence payload in XML-encoded presence information data format is expected to be produced by the Presentity/Presence User Agent (the source of the presence information) and transported to the Watchers by the presence servers or gateways without any interpretation or modification.

There are many extensions to PIDF in order to offer richer presence information, see A Data Model for Presence (RFC 4479 [32]), RPID (RFC 4480 [33]), Timed Presence Extensions (RFC 4481 [34]), and CIPID (RFC 4482 [35]).

### 2.4.6 CPL

Call Processing Language (CPL) [36] is a language for user control of Internet telephony services. It is designed to be implementable on either network servers or user agent servers. It is not tied to any particular signaling architecture or protocol. It is based on XML. Implementations of the CPL are expected to be part of both Internet telephony servers and advanced clients; as both can usefully process and direct users' calls.

As explained in [27], CPL may be used by both SIP and H.323 servers. In the case of SIP, CPL scripts can be triggered by SIP messages. CPL scripts define a decision tree which may result in signaling (proxy, redirect, reject) or non-signaling (mail, log) action. In our case, the CPL scripts are uploaded to the SIP proxy server, SER. They are stored in an external MySQL database. When an incoming INVITE comes, SER will execute the appropriate part of the CPL script based on the SIP message. The CPL script processing determines how these calls will be handled.

In [37], Alisa Devlic describes CPL extensions for context to be used for call processing services. Context parameters such as context owner, location, task, and activity are utilized as well as a context-switch which is defined and used to trigger the context-aware services based on the context information of an end user. Figure 2.9 shows the call processing logic with CPL extension scripts in SER.

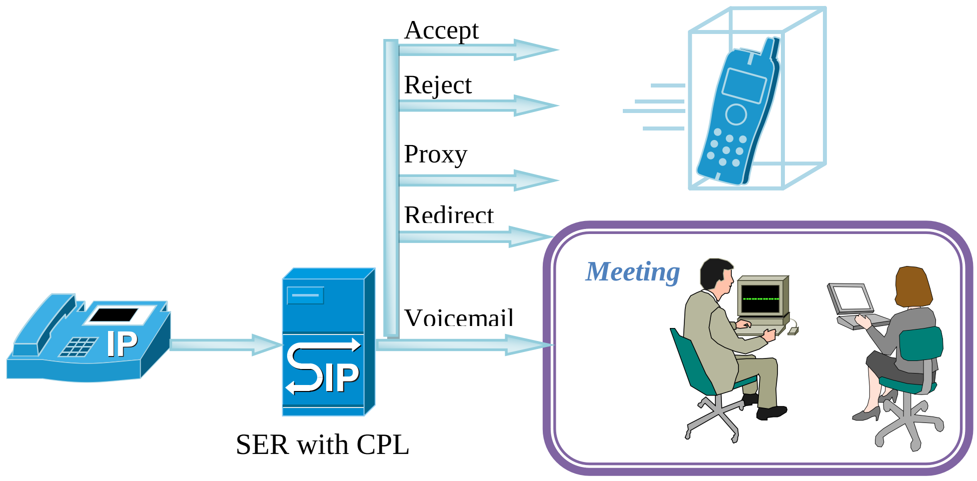


Figure 2.9 Call Processing

Below we show an example of a CPL script. In this script, an incoming call to Xueliang while he is at a meeting in an office will be switched to his voice mail based on the meeting status information detected by the occupancy detection functions leading to a context-aware system.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<cpl>
<incoming>
        <context-switch field="origin" subfield="host">
        <location url="sip:xueliang@example.com">
                <context location="Office" activity="Meeting">
                        <redirect status=”redirect” reason=”I am in a meeting.”/>
                </context>
        </location>
        </context-switch>
</incoming>
</cpl>
```

## 2.5 Related Research

Some prototypes have also been built to detect a meeting in an office environment in order to implement a smart meeting room system. We will examine the related research in this area in the following subsections.

### 2.5.1 A Conference Room Application

In [38], Conner describes a conference room application using Mica2 motes [39] to relay room occupancy data from motion sensors built into the rooms at Intel headquarters. This system utilizes in-room sensors connected to motion sensors which monitor the room occupancy status, a gateway node acts as a bridge between these sensor nodes and wireless networks, and a web application provides room occupancy information to users. This web interface allows both fixed and mobile users to view the occupancy status of the various rooms.

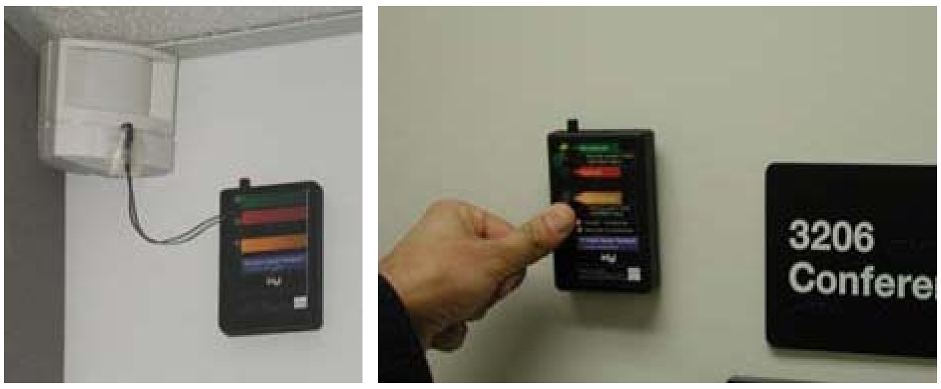


Figure 2.10 Conference Room Motion Sensor Node and Reservation Status Indicator (These figures appear here with permission from W. Steven Conner, the author of [38])

Figure 2.10 shows the motion detector nodes at the entrance to each conference room as well as a reservation status indicator which indicates current/future reservation status of the room. With this indicator, the conference room application can also provide a room reservation service through Microsoft's Outlook®. A display (at each room) shows the room's status. Communication protocols for this system consist of sensing and actuation. The sensing part delivers the occupancy information to a web server. While the actuation part provides room reservation information to the indicator attached to nodes outside of each room. A web application provides live occupancy information for all the rooms on a given building floor, as shown in Figure 2.11. A room scheduling system can also be deployed which utilizes the room occupancy information.

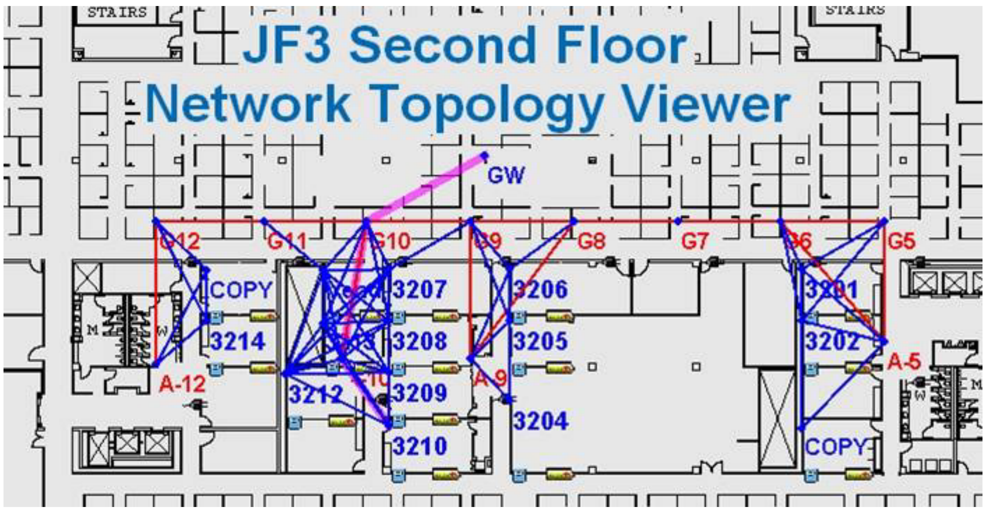


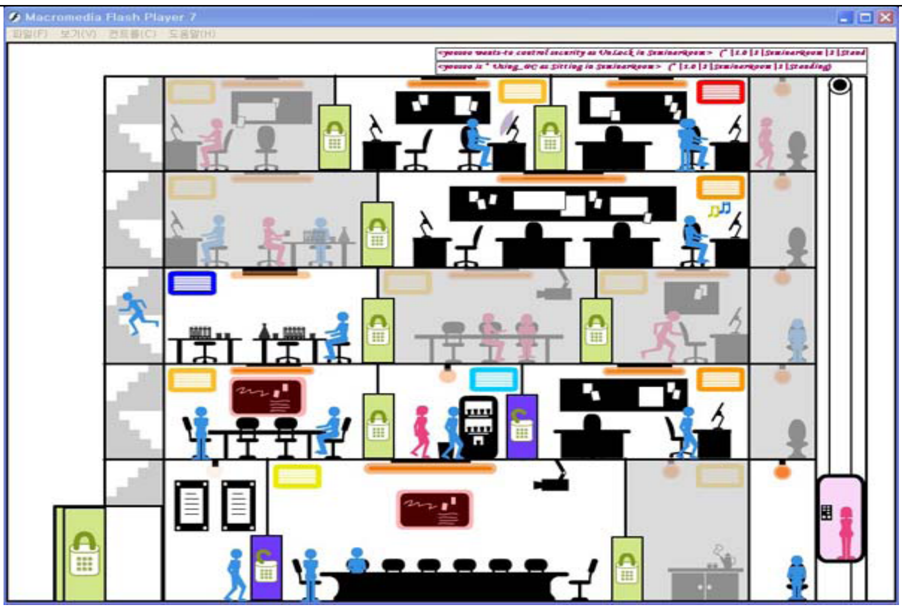
Figure 2.11 Web Page Showing Live Conference Room Occupancy and Network Topology in the Building (This figure appears here with permission from W. Steven Conner, the author of [38])

### 2.5.2 Room Occupancy Detection with Power Line Positioning

In [40], Chris Chan and Michael Onorato present a room occupancy detection system that can determine the location of the sensors nodes in a building with room-level accuracy. Their system is based on a method called "Power Line Positioning" (PLP), originally proposed in a paper by Patel, Truong, and Abowd [41]. They design their own proprietary circuits for injecting and sensing signals via the power lines. Using Panasonic ZigBee wireless communication modules [42], they obtain human presence information and send both pieces of information wirelessly to a central user interface. Another ZigBee module connected via serial port to a personal computer is used to display information through a graphical user interface.

### 2.5.3 A Large Scale Context-aware System: A Context-aware Building

In [43], Yoosoo Oh introduces a prototype of a large scale context-aware system. The prototype is a context-aware building including a number of real and simulated sensors as well as actuators. The implementation is based on three different sensors: a wearable activity sensor, environment based status sensors, and an identity, Personal Information Management (PIM) and interaction sensor on a mobile device. The wearable activity sensor as shown in Figure 2.12 (c) detects 3 axis acceleration data for a user's activity and posture. An environmental sensor, shown in Figure 2.12 (b), senses environmental data, such as sound, force, temperature, and light. They also implement an application that runs on a PC and uses the user's login and Skype states as input for an environment based status sensor. Apart from these, an application on the mobile phone acts as an identity sensor and personal information management sensor (shown in Figure 2.12 (e)). The mobile phone application also offers device controls. The sensor information can be communicated via Bluetooth from the environment to the system.



(a) The output simulator for a large building implemented in Flash(ubiBuilding)

(b)
A environment sensor

(c)
A wearable activity sensor

(d)
An extended Skype messenger acting as a status sensor

(e)
A mobile phone running the ubiMobile application

Figure 2.12 A Context-aware Building (These figures appear here with permission from Yoosoo Oh, the author of [43])

For each type of physical sensor they implement a simulated counterpart that offers the same interfaces as the real sensor and has the same function. These virtual sensors create context by clicking buttons in a graphical user interface (GUI), instead of collecting context from data detected by physical sensors. They stated that was very helpful to use these virtual sensors in the prototype period, because several physical sensor components were not yet implemented. In their prototype, the "ubiBuilding Simulator" is a simulator which can be driven by virtual or physical sensors. This output simulator is a building simulation system implemented in Macromedia Flash, see Figure 2.12 (a).

### 2.5.4 A Smart Meeting Room with Pervasive Computing Technologies

J. Wang et al. [44] implement a prototype in a typical office environment that detects the meeting's start and end by combining outputs from pressure and motion sensors installed on the chairs. The hardware and software are reasonably simple and portable. They have addressed the detection of a meeting, but they only considered pressure and motion as factors to determine if a meeting is in progress. Sometimes their system may reach an incorrect conclusion, for example assuming that two persons sitting in chairs in the library reading room for individual study are in a meeting.

In [45], Shameem Ahmed et al. present a design of a smart meeting room that not only determines the start and ending time of the meeting, but also classifies it appropriately to help make meetings more efficient and effective. They define a meeting as an activity having specific start and end times that are scheduled in time slots in the user's agenda. Some characteristics of a meeting introduced in [45] are:

a) At least two people present in the meeting
b) Most of the people occupy the chairs
c) A speaker conducts the meeting
d) Some verbal communication among people occurs
e) The silent period within a meeting does not exceed a pre-specified threshold value.

They categorize meetings into two main classes based on the conversation time. The sensing is performed by using sensors and existing software for speaker recognition. In their approach a meeting starts when some people occupy the chairs, there are some movements, and the main speaker starts to speak; a meeting ends when the meeting room is quiet and there are no movements or pressure on the chairs. They also built a simulating tool to show the results of their detection algorithm. Their approach plays a vital role not only for meeting detection, but also other closely related applications of pervasive computing.

# Chapter 3

# 3. Goals and Implementation

In this chapter, we will explain goals of this thesis and methodology for the first part. System design and implementations are also included in this chapter.

## 3.1 Goals and Methods

As explained in section 1.2, our aim was to design, develop, and deploy a meeting detector system in an office environment. Apart from the design and implementation, system testing and data analysis were performed. An evaluation based on measurements and suggested improvements was done after implementation phase. For details of the measurements and analysis see the next chapter.

The meeting detector system should provide the following functions:

a) A physical sensor entity that detects the activities of a defined space.
b) A logical sensor entity that communicates with a physical sensor and performs data gathering and processing.
c) A Presence User Agent (PUA) that distributes context data to a context server.

Furthermore the meeting detector system may be used as an input to the Meeting Room Booking System (MRBS) for later use by store the context information in a database. Therefore we should consider the interaction with other applications while designing and implementing the meeting detector system.

Using a typical project management process, the project consisted of the following phases: Product requirement specification, Design, Implementation, Testing and evaluation.

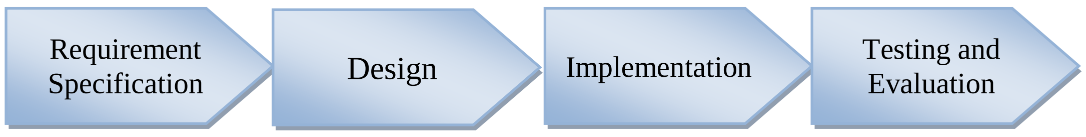


Figure 3.1 Project Management Process

At the beginning of this thesis project, several meetings with both the examiner and supervisor were held. In this phase, the requirements of this meeting detector system were discussed and defined by both the examiner and myself. After gaining a clear understanding of the project, I collected information and studied related technologies.

A literature study completed this phase and laid the foundations for the system design and implementation.

After the problem formulation process, the requirements were mapped into technical specifications. A prototype design was implemented and evaluated by iterative testing. This testing was conducted in a lab environment (using one small conference room and the hallway leading to a large open lab). The detailed hardware and software design is described in Section 3.2.

An important part of this thesis project was the evaluation of the prototype based upon system testing and data analysis. However, before discussing the evaluation further, we begin examining the design considerations with respect to the sensor system. Since the sensor provide the essential input to all of the subsequent processing. Based on the system evaluation, some improvements with regard to the system were suggested for future work.

## 3.2 Prototype Design and Implementation

In our prototype the meeting detector system has to detect the occupancy status of a defined zone and distribute the resulting context data to a context server. We integrate an occupancy sensor system with a SIP Express Router (SER) based context-aware architecture. An overview of this system is shown in Figure 3.2. The occupancy sensor system generates context information as input to SIP Express Router which works as a context server.

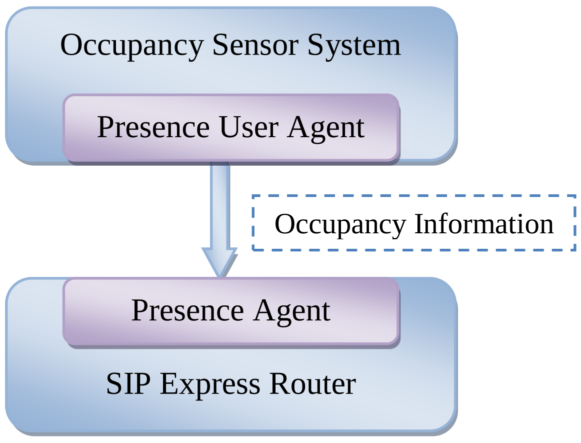


Figure 3.2 System Architecture

We will describe the design of prototype specifically in the sections which follow. We start with the sensor itself, as the characteristics of this data source affect all of the subsequent parts of the system.

### 3.2.1 Sensor System Setup

In this section, we will explain the setup and configuration of our sensor system on both hardware and software.

#### *3.2.1.1 Hardware Description*

##### *(a) Velleman PIM Intrusion Indoor Detector, HAA52*

For the sensor hardware, we choose the Velleman PIR Intrusion Detector [46], HAA52, which is a thermal detector as explained in Section 2.3.1. The HAA52 is a reliable low-cost detector designed for general use.

It has the following features and specifications:

- Dual-element pyroelectric sensor
- Programmable pulse counter
- 12° vertical angle adjustment
- RFI protection

- Wide operating input voltage range: DC 8-16V
- Suitable for surface and corner mounting
- Detection pattern: wide angle, 90° horizontal, maximum 42 beams in 3 layers
- LED indicator: walk-test indication ON or OFF (this can be enabled or disabled)
- Alarm activation delay: 2 to 3 seconds

Based on these features, it seems very suitable for our sensor system. It can be placed at the boundary of an area to monitor the movements across the border. In our prototype we plan to deploy the HAA52 detector at the entrance door of a meeting room in order to monitor the entrance and exit activities - as this will enable us to compute the occupancy status of the area. Note that an essential feature of this sensor is the dual-element pyroelectric sensor, as this enables us to determine if someone is entering or leaving the room.

### *(b) Velleman USB Experiment Interface Board, K8055*

For data collection, we choose the Velleman USB Experiment Interface Board [47], K8055. It is a small USB attached board with several I/O ports. It is powered via the USB interface. A picture of this interface board is shown in Figure 3.3.

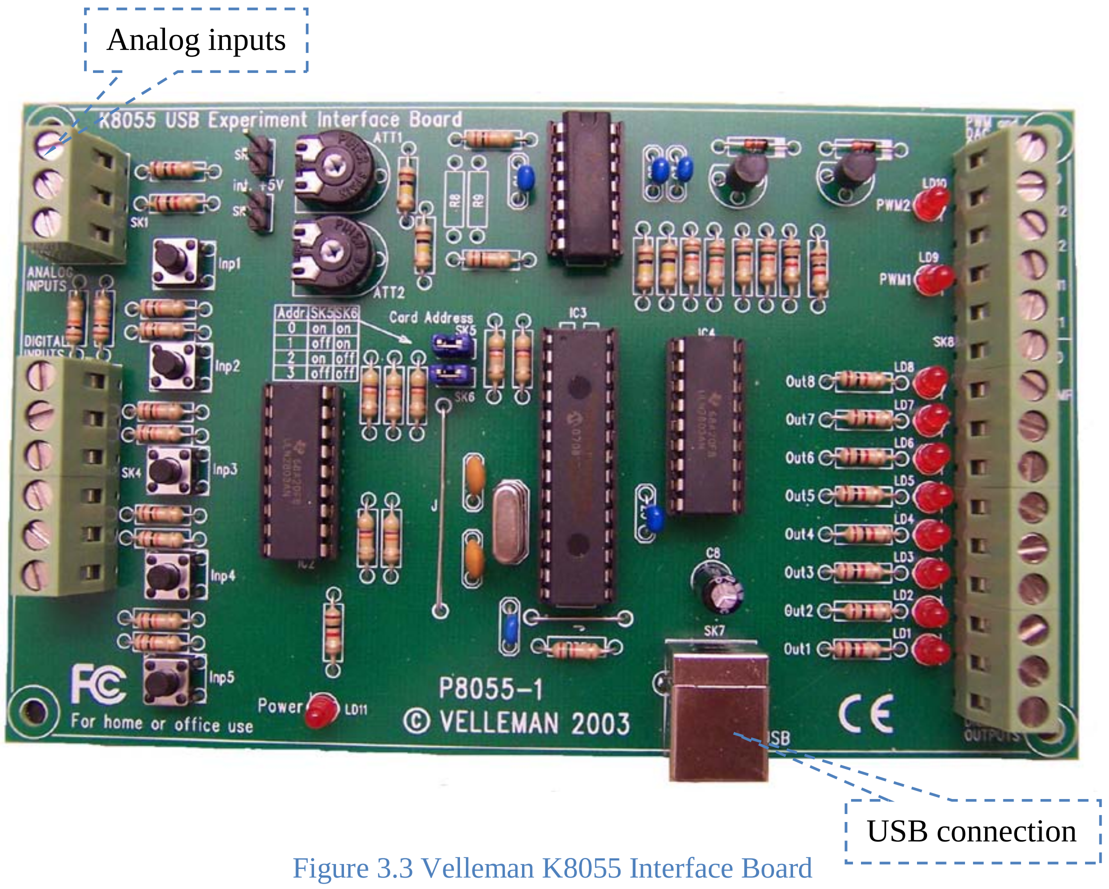


Figure 3.3 Velleman K8055 Interface Board

Below is a summary of the features and specifications of the Velleman K8055 interface board:

- Power supply through USB: approx. 70mA
- 5 digital input channels (0=ground, 1=open), and test buttons are provided on board
- 8 digital output channels
- 2 analog inputs with 8 bit resolution & attenuation with optional amplification (internal test +5V provided)
- 2 analog outputs with 8 bit resolution
    - 0 to 5V, output resistance 1K5
    - PWM 0 to 100% open collector outputs: maximum 100mA/40V (on board LED indication)
- 2 hardware counters connected to the 2 first digital inputs with customizable debounce times
- Diagnostic software and communication Dynamic Link Library for the Microsoft Windows platform
- General conversation time: 20ms per command
- The numbers of inputs/outputs can be further expanded by connecting more (up to maximum of four) cards to the PC's USB connectors

While this board is overkill for our application, it provides all the features necessary to collect data from two HAA52 detectors. There is an illustrated assembly manual available at [48]. This K8055 Interface board is easy to use and there existed C source code for use on a Linux platform (see section 3.2.1.2).

Actually we intended to use a SmartBadge at the begging of this project. It was designed by Mark T. Smith. More information about this badge can be found in [49]. It could fulfill the implementation for us with some programming on it. However, after some investigation and comparison, we found the K8055 interface board matched our requirements very much – it has 2 anolog inputs with 8 bit resolution for sampling which is enough for us and it is easy to be deployed with the physical sensor entity. So we decided to buy and use this board for our system.

### *(c) Dell OptiPlex GX620 Desktop PC*

For data processing and programming, we use a Dell OptiPlex GX620 desktop PC. The specifications of this OptiPlex GX620 PC we used are:

- Intel® Pentium® D Processor with Dual Core Technology with 2.8GHz CPU
- 2GB of Dual Channel DDR2 memory
- 250GB of Serial ATA II (SATA II) hard drive
- ATI Radeon X600 with 256MB of memory
- Dual VGA displays
- Dual Operating Systems: openSUSE Linux and Microsoft's Windows XP

In this thesis, we used this OptiPlex GX620 PC as the main development and operating platform. The hardware system is configured as Figure 3.4.

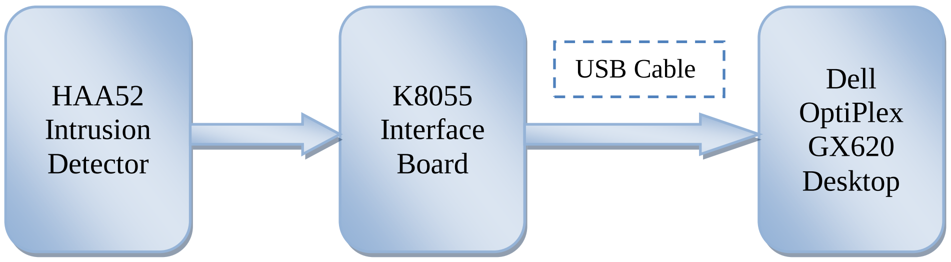
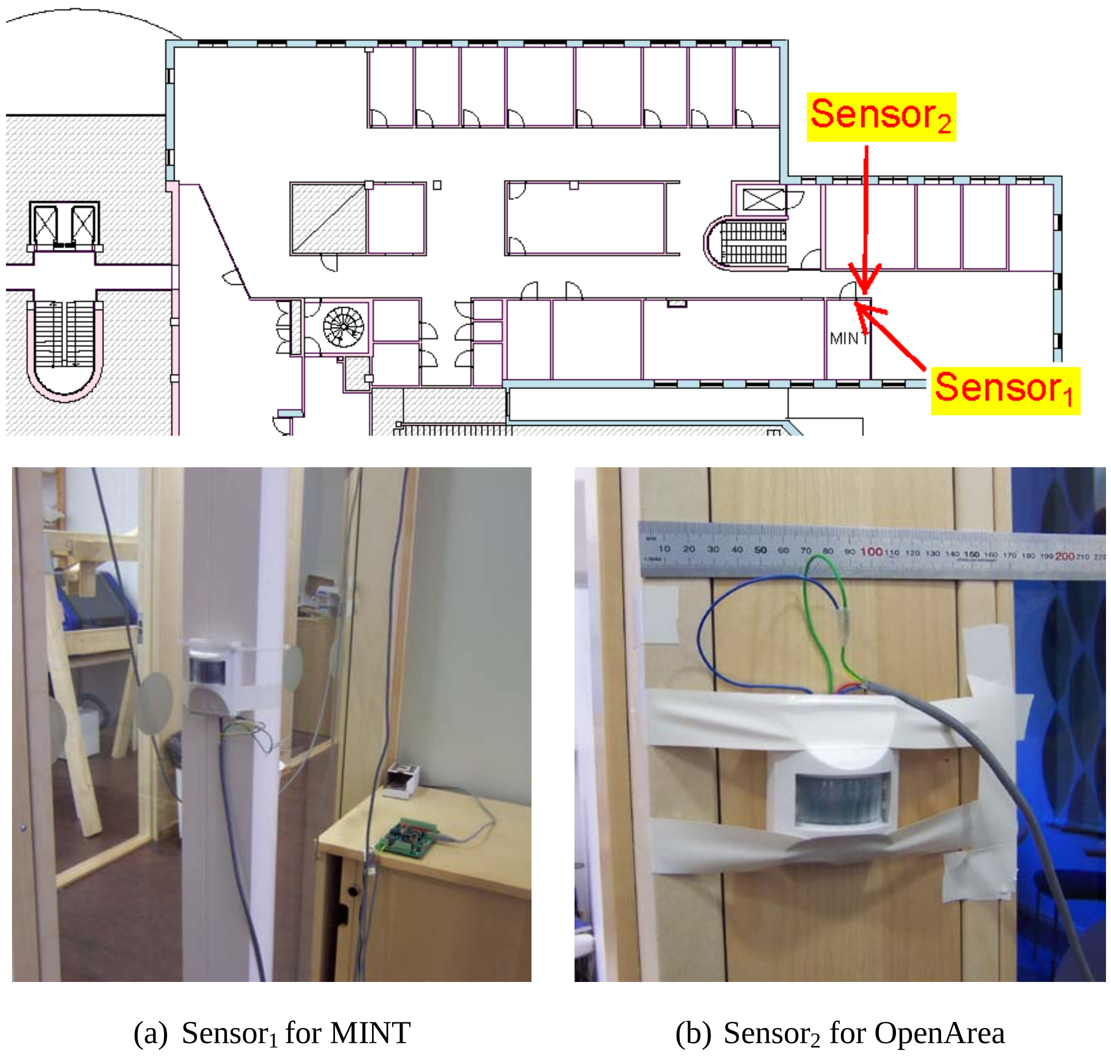


Figure 3.4 Sensor System Hardware Setup

The HAA52 detector is attached to a pillar at the entrance door to the meeting room (See Figure 3.5) and connected with a K8055 interface board. At the same time, the K8055 interface board is connected to the Dell GX620 PC via USB cable. So the sensor data from the sensor is gathered and converted by the K8055 interface board and finally transmitted to the PC.



(a) Sensor$_1$ for MINT (b) Sensor$_2$ for OpenArea

Figure 3.5 Sensor Placement

For testing we placed one HAA52 detector in a meeting room "MINT" for initial testing and measurement. Then another HAA52 detector was deployed in the corridor to monitor the "OpenArea" (See figure 3.5). These two detectors were both connected to the K8055 interface board as input 1 and input 2. Later on more detectors and interface boards could be deployed for the other conference rooms in our environment.

#### *3.2.1.2 Software Setup*

For the K8055 interface board, Velleman provides a Dynamic-link library (DLL) which contains all the routines needed to write your own programs to control this interface board. The functions provided by the DLL are pretty straight forward and self-explanatory. The library itself is based on the DLL developed for the Windows platform. Sample PC software is provided in the folder K8055_VM110 USB board on the Velleman software CD. We intend to install the demonstration software on a Windows machine for initial functional testing of the K8055 interface board.

The included demonstration software makes it easy to experiment. The jumpers SK5 and SK6 are used for address selection (open = 1, closed = 0). In other words, we can choose the interface board number by jumpering. Up to four boards can be connected at the same time. In our case, we jumpered SK5 and SK6 in order to select address 0 for this K8055 interface board. The address selection details are shown in Table 3.1.

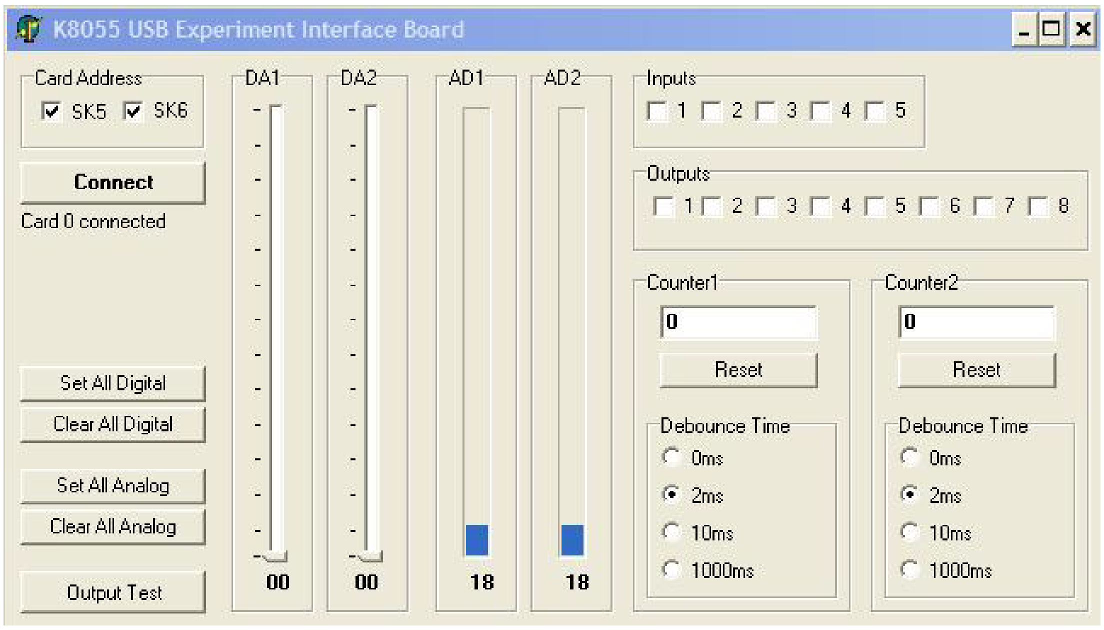
Table 3.1 Address Selection

| SK5 | SK6 | Address |
|---|---|---|
| ON | ON | 0 |
| OFF | ON | 1 |
| ON | OFF | 2 |
| OFF | OFF | 3 |

Then we followed the test procedures in the manual as follows:

- Connect the USB cable to the board
- LED LD3 "Power" lights up if the board is properly connected
- After startup LD8 (output 8) will flash momentarily to indicate that the circuit works as it should
- Start the program "K8055_Demo.exe"
- The software displays a graphical user interface as shown in figure 3.6.
- Press the "connect" button to connect the K8055 with the PC
- The message "Card 0 connected" is displayed to indicate the connection is successful. See Figure 3.6.

Now we can generate some binary inputs by pushing the 5 digital input buttons. Input is generally "high" (1), while connection to GND makes the input "low" (0). We can also use the internal analog voltage to provide an input via potentiometers RV1 & RV2 on the board. In our application, the analog inputs are used to attach to our sensor, which may be from a temperature sensor, a potentiometer, thermal detector, … .



K8055 USB Experiment Interface Board

Card Address: [✓ SK5] [✓ SK6]

Card 0 connected

DA1: 00
DA2: 00
AD1: 18
AD2: 18

Inputs: 1 2 3 4 5

Outputs: 1 2 3 4 5 6 7 8

Counter1: 0
Debounce Time: 0ms [✓ 2ms] 10ms 1000ms

Counter2: 0
Debounce Time: 0ms [✓ 2ms] 10ms 1000ms

Figure 3.6 K8055 USB Interface Board Demo

Here we connect the pyroelectric HAA52 detector to the K8055 interface board. Details of how to connect directly to the pyroelectric sensors are contained in Daniel Hübinette's earlier thesis [1]. The connecting two of these detectors to the board results in the display shown in Figure 3.6. The scroll bars AD1 & AD2 on the screen show the digital value (0 - 255) from the analog to digital converter. To convert this value to an actual voltage, we have to connected a know voltage source to the device and set the scaling appropriately (using a potentiometer). Then it is possible to translate each value to a voltage. However, we are not really concerned with exact voltages, but rather the change in voltage as someone walks by this sensor. This simple testing enabled us to see that the interface board was working and that it might provide the interface to the sensor which we desired. The next step was to write a program that could make some measurements to see if we could sample the analog signal from the sensor at an adequate sampling rate.

Before developing our own application program, we looked to see if there was an existing library that could be used with Linux (thus we could develop all of the software and run it on a Linux system). Our preference was to find an open source driver for this board for Linux, because we could then integrate this application with other Linux programs. With this thought in mind, we searched for a Velleman K8055 device driver for Linux. After some searching, testing, and comparison, a Linux

K8055 library provided by Sven Lindberg was chosen. This software is available at [50]. It is under GPL license which suited our requirements for an open source solution. A command line tool and a manual page are included with this software. Lindberg developed his library to replace all other half-complete software for the K8055 board under Linux. We used it to get control of the K8055 interface board. It should be noted that it has the same functions as described in Velleman's DLL user manual, thus it is easy for a user to port programs between Windows and Linux (at least with respect to controlling this interface board). For example, we can use this library to read all the inputs, set digital/analogue outputs, or set debounce time.

We also have noted that a Linux version of Graphical User Interface (GUI) based library is being developed [51]. It is a C/C++ program based on libk8055 USB board library and wxWidgets GUI library. This could have been used to test the hardware (much as we did with the demonstration applications provided by the hardware vendor). In this thesis project, we did not use this GUI program, but it might be useful for others.

For the development platform we mainly used openSUSE 10.3 Linux OS in this thesis project. OpenSUSE 10.3 was released with Linux kernel version 2.6.22.5 and GNU Compiler Collection (GCC) version 4.2.1. We updated the Linux kernel to version 2.6.22.17 for more stable use.

The software environment we used consists of:

- openSUSE 10.3 with kernel 2.6.22.17
- GCC 4.2.1
    - a set of compilers produced for various programming languages (C, C++, Java, Fortran …) by the GNU Project
- libusb 0.1.12
    - an open source library that allows you to communicate with USB devices from user space regardless of OS
- Linux K8055 library
    - a Linux library with functions for Velleman K8055 interface board

The Linux K8055 library is available at [50] with the file name "libk8055.0.2.tar.gz". We downloaded it and installed it following the instructions:

Extract it to the folder in path: */usr/src/k8055*

Then execute these commands to compile and install (where "ccsmoto" is the local host name)

```
ccsmoto:/usr/src/k8055 #    make all
ccsmoto:/usr/src/k8055 #    make install
```

The default installed paths are:

*/usr/local/bin, /usr/local/lib, /usr/local/include, and /usr/local/man/man1/*

Note that there is a manual page which can be read by invoking

*ccsmoto:/usr/src/k8055 #    man k8055*

Our own program (readsignal.c) can use the k8055 library with this argument:

*ccsmoto:/usr/src/k8055 #    gcc -Wall -lusb -L/usr/lib -lm -lk8055 readsignal.c*

We developed our own program to read signals from the Velleman K8055 interface board using the include file k8055.h and the k8055 library. This program reads the context data from the board and sends the sensor values to another logical entity (implemented on this same PC - but it could be placed elsewhere on the network) via a UDP socket. The processing is purposely split in this way to facilitate the implementation later of a network attached sensor (i.e., a pyroelectric sensor node could be designed and built as an Ethernet attached device - getting its power via Power over Ethernet).The "logical.c" program takes charge of the context data processing and provides this data to the context server (SIP Express Router).

The Linux K8055 library also comes with a command line application that can be used for testing.

**Syntax:** k8055 [-p:(number)] [-d:(value)] [-a1:(value)] [-a2:(value)] [-num:(number) [-delay:(number)]] [-dbt1:(value)] [-dbt2:(value)] [-reset1] [-reset2] [-debug]

This k8055 command can be run by root or any other user with permission to use the USB. A more detailed description of these options can be found in manual page for the k8055. There are some bugs with k8055 library. The 0 - 7450 ms is split along an exponential scale in 255 sections according to Lindberg's report. The debounce timer values are not accurate by the millisecond despite Lindberg stating that it works within +-4% accuracy of the actual set time. However, it is more precise than Velleman's DLL

The main.c program in the source file of libk8055 implements the above k8055 command. Initially we adapted this program (with a new file name "readtest.c") and used it to collect some analog data from channel 1 (input 1) and to write this data to a log file for some initial testing. This program proved the driver worked correctly with this K8055 interface board. This test program also allowed us to understand the maximum rate at which we could sample the analog data. The successful results of this testing enabled us to conclude that we could continue our implementation of the occupancy detector. The readtest.c program listing can be found in Appendix A.

### 3.2.2 Detection Approach

#### *3.2.2.1 Interaction between Physical and Logical Sensor Entities*

We need to collect the sensor data for the logical sensor. This process begins by reading the value from the analog channel on the K8055 interface board. Then the sensor data should be sent to the logical entity to do movement detection and analysis. A structure for the interaction between physical and logical sensor entities is shown as Figure 3.7.

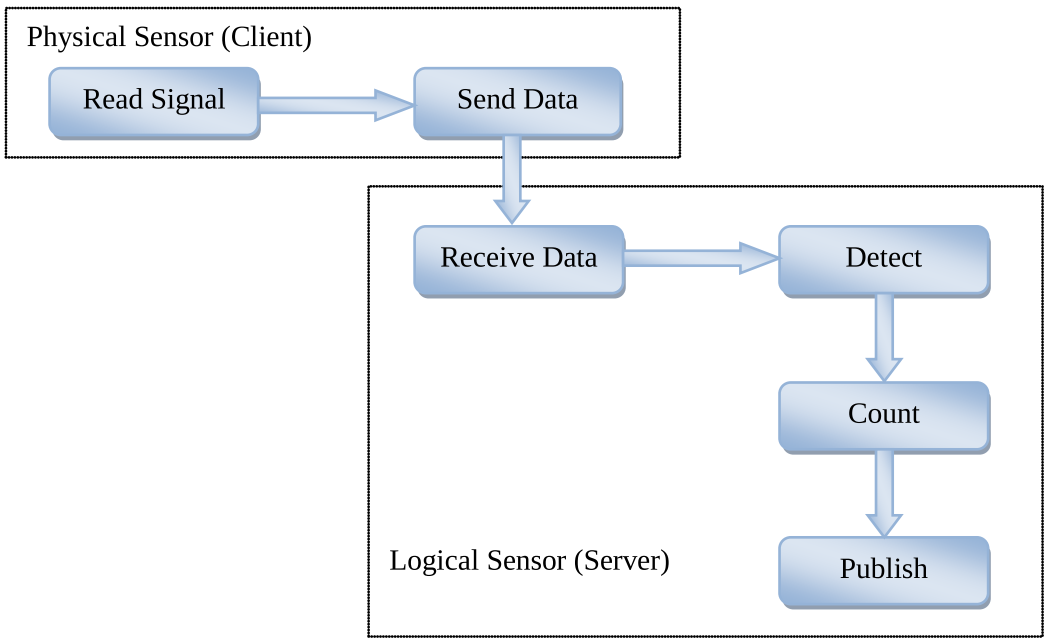


Figure 3.7 Physical and Logical Sensor Software Structure

In our work, the physical and logical sensor entities reside in the same computer which is running openSUSE Linux. A client-server model between the physical and logical sensor entities can be made to transmit sensor data. The system has a startup phase to do initialization of the hardware and software system. The software both for the physical sensor and logical sensor is written in the C programming language. The physical sensor entity reads the signal waveform data from the sensor and sends it to the logical sensor entity via a UDP client/server program. The logical sensor entity receives the stream of data and executes a detection algorithm to compare the data with the earlier state in order to indicate whether movement past the detector occurred. If so, then the occupancy status will change. This context information can be sent to the intended entity as a publish message.

As we indicated before, a UDP socket for client/server communication is used. Using UDP, programs on networked computers can send short messages sometimes known as datagrams (using Datagram Sockets) to one another. Sensor data are sent from the physical sensor entity to the logical sensor entity as UDP packets. The generic UDP packet structure [52] is shown as Table 3.2.

Table 3.2 User Datagram Header Format

| Bit 0 - 16 | 16 - 32 |
|---|---|
| Source Port | Destination Port |
| Length | Checksum |
| Data | |

UDP provides two services that are not provided by the IP layer. It provides port numbers to multiplex & de-multiplex packets and, optionally, a checksum capability to verify that the data have arrived intact. The source Port indicates the UDP port of the sending process. The destination is used to determine the process to which the UDP datagram should be delivered at the destination. The 16-bit checksum field is used for error-checking of both the header and data. A 16-bit length field that specifies the length in bytes of the entire datagram: header and data. The minimum length is 8 bytes since that is the length of the header. The field size sets a theoretical upper limit of 65535 bytes for the data carried by a single UDP datagram.

There is a relation between payload size and packetization delay of the sensor's samples. A larger UDP payload size results in the greater packetization delay. Another issue to be considered is the rate at which user data is generated in the physical sensor entity. This data rate is related to the numbers of bytes read from the interface board per command. Reading a larger number of samples per command reduces the time needed to collect data for placing in the payload and results in the lower packetization delay. In the prototype the maximum raw data sample rate is much lower than the capacity of the network. Actually we were using a USB connection between the sensor hardware and the physical entity. So the bottleneck was not the network traffic problem.

UDP ports enable application-to-application communication. The port field is a 16 bit value, allowing for port numbers to range between 0 and 65535. The private ports are those from 49152 to 65535. We choose 49152 as a static UDP port as the destination port for the logical entity. It is statically defined in the "address.h" header file (See Appendix D) which contains some other server addresses and ports.

A small sample of the source code to read data from the K8055 interface board is shown below. It reads an analog voltage separately from channel 1 and channel 2. The sample data can be printed for testing and debugging (if necessary). Two buffers are used for temporary storage.

```
for (i=0; i < Number_of_Bytes_to_read; i++) {
            sample=ReadAnalogChannel(1L);
            sample2=ReadAnalogChannel(2L);
            bigBuffer[i]=(unsigned int)(sample & 0xff);
            bigBuffer2[i]=(unsigned int)(sample2 & 0xff);
}
bigBuffer[Number_of_Bytes_to_read]=0;
bigBuffer[Number_of_Bytes_to_read+1]=0;
bigBuffer2[Number_of_Bytes_to_read]=1;
bigBuffer2[Number_of_Bytes_to_read+1]=0;
```

In order to distinguish the two buffers for the logical entity, we add a two byte value into the buffer to indicate which sensor the data was read from. In this way multiple detectors could be used and easily identified based on these bytes.

A UDP client socket is created in the physical sensor program to send data to the logical sensor entity. The following lines show parts of the socket code on client side.

```
if ((sendto(client_socket_fd,
            bigBuffer,
            Number_of_Bytes_to_read+2,
            sendto_flags,
            (struct sockaddr*)&server_addr,
            sizeof(server_addr)
        )
      ) == -1) {
    perror("Unable to send to socket");
    close(client_socket_fd);
    exit(1);
}
```

The full program for the physical entity is named "readsignal.c" and can be found in Appendix B. The physical entity code can be compiled by invoking

```
ccsmoto:/usr/src/k8055 # gcc -o readsignal -lusb -L/usr/lib -lm -lk8055 readsignal.c
```

Then an executable file readsignal will be created. To manually run this program you can simply type

```
ccsmoto:/usr/src/k8055 # ./readsignal
```

When the program is executed the Velleman interface board is found and sensor data sent to the logical entity via a UDP socket. When the program starts, the following output will be generated by the library routines:

```
Velleman Device Found @ Address 011 Vendor 0x010cf Product ID 0x05500
Got driver name: usbfs
Disconnected OS driver: No error
Found interface 0
Took over the device
```

#### *3.2.2.2 Detection Algorithm/Methodology*

The purpose of the detection algorithm is to know the occupancy status of an area. This section describes how to process the raw data received from the physical entity in order to determine the area occupancy status. The sensor raw data is collected from the physical entity via a UDP server socket. Two approaches can be used to analyze and process the data. We mainly use the same algorithms as Daniel did in his earlier thesis project. One is voltage threshold state based and the other is correlation based detection algorithm.

##### *a. State based model*

As we know from the testing, the voltage output from the detector changes when someone passing by the detector as shown in Figure 3.8. Examining the analog voltage waveform of the detector with an oscilloscope, it has a voltage range between zero and two volts approximately. Five zones are defined: Top Zone, Middle of Top Zone, Normal Zone, Middle of Bottom Zone, and Bottom Zone. The zone boundaries can be measured and determined from analysis of the sensor raw data.

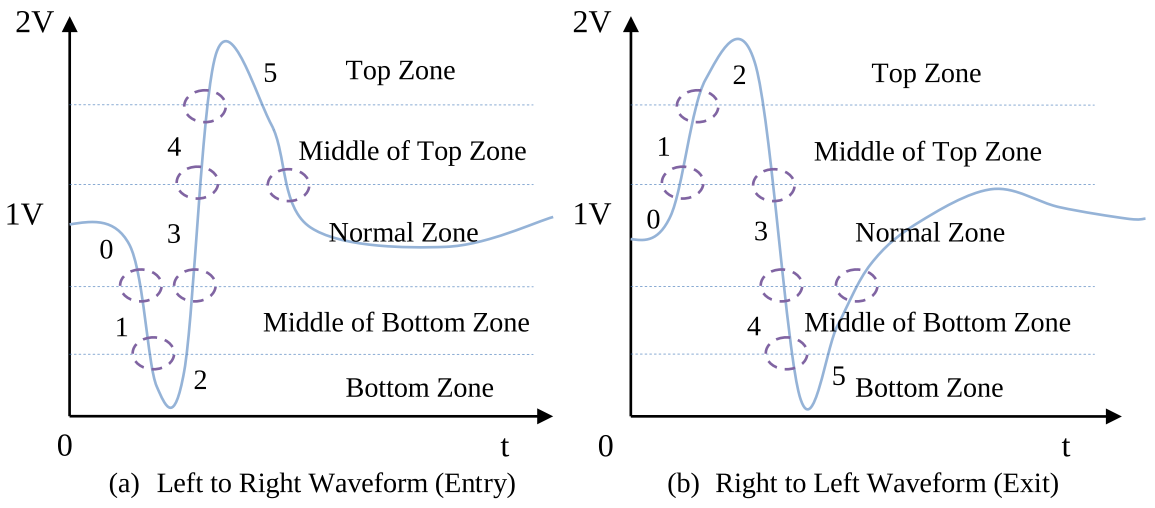


Figure 3.8 State based Detection Framework

The signal waveform can be divided into different states based upon the signal crossing the boundaries. A set of state transitions indicate the detection of a change in occupancy using a state based detection algorithm. The voltage range near one volt is defined as state 0. When the voltage goes across the threshold (either higher or lower), it changes to the next state. Once the states from 0 to 5 have been traversed in a particular order, a detection of entry or exit is made. Then the occupancy status of the area is changed accordingly. Note that a timer is used to reset the state to 0 if a defined time period is exceeded without a state change being triggered. So the stable state is state 0. States 0 to 5 correspond to the states of receiving data, detecting, and comparing. State S before 0 stands for the Startup state for initializing to receive data.

State C after 5 corresponds to the count state to update the count of people in the area. This can be considered as a state machine.

This state based detection algorithm can only detect one person passing through the boundary at a time. The algorithm state can be reset when the numbers of entries or exits exceed a predefined value. The detailed state based detection algorithm is described in the logical sensor code, which can be found in Appendix C.

### b. *Correlation based model*

A correlation based detection approach can be used for data processing and detection computing. In [53], correlation is introduced as it indicates the strength and direction of a linear relationship between two random variables in probability theory and statistics. The main result of a correlation is called the correlation coefficient (denoted by r). The best known linear correlation coefficient is sometimes referred to as the Pearson product moment correlation coefficient. This is computed by dividing the covariance of the two variables by the product of their standard deviations.

The mathematical formula for computing r is:

$$r = \frac{Co\operatorname{var} XY}{stdevX \cdot stdevY} = \frac{n\sum XY - \left(\sum X\right)\left(\sum Y\right)}{\sqrt{n\sum X^2 - \left(\sum X\right)^2}\sqrt{n\sum Y^2 - \left(\sum Y\right)^2}}$$

Where n is the number of pairs of data.

The correlation value ranges between -1 and 1$(-1 \leq r \leq +1)$. The + and – signs are used for positive correlation and negative correlation, respectively. The closer the coefficient is to either -1 or 1, the stronger the correlation (or in the negative case, the anti-correlation) between the variables.

**Positive correlation:** If the correlation value r is positive, this means that as one variable gets larger the other gets larger as well. If X and Y have a strong positive correlation, r is close to +1. An r value of exactly 1 indicates a perfect positive fit.

**Negative correlation:** If r is negative it means that as values for X increase, values for Y decrease (usually called "inverse" correlation). If X and Y have a strong negative correlation, r is close to -1.

**No correlation:** If r is close to 0, this means there is no relationship between the two variables.

Note that a correlation with r greater than 0.8 is generally described as strong, whereas a correlation less than 0.5 is generally described as weak.

Basically, we can use correlation to get the similarity between two series of data. In other words, we can compare the real time sensor data (Y) with a template sensor data (X). In the correlation based detection system, template sensor data is collected and chosen in advance. It is reversed in time and is compared with the incoming real time sensor data (the reversal in time is to facilitate the incremental computation of the correlation). Then we compute the correlation coefficient as each sample (or group of samples) arrives to learn the relationship between the real time sensor data and template sensor data. The correlation coefficient r is computed at each discrete time step. A threshold level is set to indicate when the real time sensor data is close enough to the chosen template data to trigger an entrance or exit detection. The threshold of correlation coefficient r should be determined based on sufficient measurements of detection in order to guarantee the desired accuracy. In this thesis project, the threshold was set at r = 0.8.

Correlation based detection is widely used since it is very suitable when dealing with irregular waveforms. It offers greater accuracy than a state based algorithm. For multiple detectors, we can use the same template sensor data if the room environment is similar. Otherwise we can also have multiple series of template data for multiple detectors. In our detection program, a correlation based algorithm is used to monitor the room occupancy status and notify the context server via publish messages when the occupancy status changes. The code for correlation based detection is described in Appendix C.

In this project, both detection algorithms are written in the same logical code. Thus they can execute at the same time. The program can be compiled by this command:

```
ccsmoto:/usr/src/k8055 # gcc -o logical -lusb -L/usr/lib -lm -lk8055 logical.c
```

Then an executable file can be used to receive the sensor data, compute detection, and publish the occupancy information to context server.

```
ccsmoto:/usr/src/k8055 # ./logical
```

In this program, the specific numbers of persons in the area can be calculated based on the detections of the number of entries and exits. Then the occupancy status (the number of persons in the area) can be obtained. Therefore we can infer whether there is a meeting in progress or not. The room status is quantized as defined as Table 3.3.

Table 3.3 Room Status Definition

| Persons in the area | Room status |
|---|---|
| 0 | Empty |
| 1 | Individual |
| $\geq 2$ | Meeting |

Furthermore, meetings can be categorized according to the numbers of people in the area. For example, a small meeting (2-3 persons), medium sized meeting (3-4 persons), and large meeting (more than 5 persons). This information may be later used in the room scheduling system to monitor the meeting room utilization (i.e., to understand if small meetings are taking place in large rooms or not).

The room occupancy information is distributed to a SIP proxy server using the SIP/SIMPLE protocol. The context information is encoded as a PIDF formatted document. These protocols and standards were described in Sections 2.4.3, 2.4.4, and 2.4.5.

### 3.2.3 Publish Context Information to a SIP Proxy

The room occupancy information can be used by our context server. For the context server, we use the SIP Express Router (SER) as a SIP proxy server. The context module in SER should be able to handle publish, subscribe, and notify messages. In our case, it needs to receive the publish messages from the logical sensor entity and parse them to extract the relevant context data from these messages and store it in database for other context applications use. Mohammad Z. Eslami implemented a presence server in Ubuntu Linux using a under development version of SER with the PA module in his thesis project in 2007. The source code he used was ser-0.10.99-dev35-pa-4.2_src.tar.gz from the "iptel.org" website. He modified the source code and added some new modules to handle different kinds of events, such as location. However, we decided to start from the latest SER source code and adapt it for our use.

#### *3.2.3.1 Installation and Configuration of SER*

In order to handle the occupancy information from our meeting detector system, I decide to implement my own context server based on SER. An openSUSE Linux (with the host name "ccsleft") has been used as the platform. The configuration of this machine is roughly the same as the machine described earlier in section 3.2.1.1.

SER is under continuous development and many new features have been added since the earlier thesis project. SER 2.0 [54] has been thoroughly tested and has been available since August 6, 2008 from iptel.org's FTP server. It appears to be a stable release with the source code file name ser-2.0.0_src.tar.gz. For this project I decided to use SER 2.0 which includes a PA module.

In order to build SER from its sources, we need the following:

- GCC: version 3.1 or higher recommended (we use gcc 4.2.1)
- bison: GNU Project parser generator (we use GNU Bison 2.3)
- GNU make: version 3.79 or newer (we use GNU Make 3.81)
- GNU tar
- GNU installs

Some additional packages may be needed to build all the modules:

- libmysqlclient & libz (zlib) for MySQL support (the mysql module)
- libexpat for jabber gateway support (the jabber module)
- libxml2 for the cpl-c (CPL support), pa (presence) and xmlrpc modules
- libradiusclient-ng (> 5.0) for radius support (the acc_radius, auth_radius, avp_radius, and uri_radius modules)s
- libssl for SSL/TLS support (TLS module)

Since most of the software and libraries are included in openSUSE 10.3 Linux in its default installation, it is not too complicated to install SER. However, configuration of this software to suit our purpose and debugging each of the trial configurations required a lot of time.

We downloaded the source code from iptel.org's website and installed it following the instructions below:

Download it and unpack it to the directory: */usr/src/*

> *ccsleft:/usr/src/ # tar xzf ser-2.0.0_src.tar.gz*

Then cd to the main directory of ser: */usr/src/ser-2.0.0*

> *ccsleft:/usr/src/ # cd ser-2.0.0*

To compile core and the basic set of modules (standard modules), we can just execute:

> *ccsleft:/usr/src/ser-2.0.0 # make all*

By default modules that require external libraries or that are considered to be "experimental" will not be built. However, we can choose to include specific modules when compiling from the source package.

To compile standard and standard-dep modules with dependencies that must be satisfied for compilation, the command line shown below should be used:

> *ccsleft:/usr/src/ser-2.0.0 # make group_include="standard standard-dep" all*

This will add the standard modules with dependencies (this requires installing the mysql, mysql-devel, libcurl, libcurl-devel, libxml2, and libxml2-devel packages). The command "make print-modules" shows which modules are build by default. Here all modules that will be in included and excluded will be listed.

Then execute (Make sure you use the same group_include here as compilation):

*ccsleft:/usr/src/ser-2.0.0 # make group_include="standard standard-dep" install*

Finally SER is successfully installed in /usr/local (its default location). See the INSTALL file in the main directory of SER for detailed information on installation notes.

After installation of SER, it needs to be configured up and running. A configuration file is used. Usually it is located in the directory */usr/local/etc/ser* with the name "ser.cfg". In this directory some sample configuration files are provided for a variety of uses. One can create a new configuration file or modify the default ser.cfg. As we want to load some specific modules and create a specific configuration, we modified the default "ser.cfg" file for our purposes. An example of a ser.cfg we used can be found in Appendix E.

To start SER, we can use:

```
ccsleft: # /usr/local/sbin/ser –E
```

It may be necessary to set LD_LIBRARY_PATH before startup by this command:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/ser/modules:/usr/local/lib/ser
```

For persistent data storage, SER can be configured to support MySQL. We used this command to create an initial MySQL database:

```
ccsleft: # /usr/src/ser-2.0.0/scripts/mysql/ser_mysql.sh create
```

Note that a new occupancy table was added in the database. Thus the my_create.sql file was edited to add this support for occupancy information. The SQL code to create the additional table is:

```
INSERT INTO version (table_name, table_version) values
('presentity_occupancy','5');
CREATE TABLE presentity_occupancy (
    sensorid INT(3) NOT NULL,
    pres_id VARCHAR(64) NOT NULL,
    basic VARCHAR(64) NOT NULL,
    occupancy VARCHAR(64) NOT NULL,
    note VARCHAR(64) NOT NULL,
    contact VARCHAR(64) NOT NULL,
    tupleid VARCHAR(64) NOT NULL,
    UNIQUE KEY presid_index (pres_id, tupleid)
);
```

Another command can be used to convert the SER database to support the new structures (if SER was previously installed):

```
ccsleft: # /usr/src/ser-2.0.0/scripts/mysql/ser_mysql.sh reinstall
```

In addition to modifying the configuration of the MySQL database to support the SER database and tables, the ser.cfg file must also be edited to support the modified database. After creating the database we need to specify a domain name and a user name in the system for applications use. It is recommended in the SER documentation to use the Serctl command line interface [55] for controlling SER. Serctl is a command line utility for SER administration. To install Serctl, requires some additional packages (python, python-mysqldb). After the installation of Serctl, we used it to control SER.

First we add a domain name (in this case we use the IP address of this SER server as the domain name):

```
$ ser_domain add 130.237.15.238 ser
```

Then we add a user "xueliang" with password "heslo" using the following three steps. First, we add the user, then assign one or more URIs to this user, and finally add credentials so that the user can register.

```
$ ser_user add xueliang
$ ser_uri add xueliang xueliang@ser
$ ser_cred add xueliang xueliang 130.237.15.238 ser helso
```

At this point, SER will be up and running with database support. For our context server, we needed to adapt the PA module for our occupancy information since the presence event is supported by default. Unless this adaptation is made, publishing any other event type simply generates an "unsupported event message". The details of this are described in the next section.

#### 3.2.3.2 Implementation of SER Modules to Handle Occupancy Event

We introduced SER in section 2.4.3. In order to extend SER to support our new events required understanding the SER architecture and configuration file in depth. SER is built around a processing core that receives SIP messages and provides the basic functionality need for handling SIP messages [56]. SER has a modular architecture and most of the functionality is provided through SER modules. The ser.cfg file controls which of these modules are loaded and module specific variables are defined to control the configuration and behavior of each of these modules.

Several modules are basic parts of "presence" support. These modules are: [57]

Table 3.4 SER Presence Modules

| | |
|---|---|
| **PA** | The PA module allows SER to act as a presence server. Its main function is to process subscriptions to presence state of standalone users and to process presence state publications for them. |
| **RLS** | The resource list server gets presence information for standalone users from internal queries to PA module or remote presence server queries and builds them together into list notifications. |
| **Presence_b2b** | This module can be used to subscribe to presence state on remote server. |
| **XCAP** | This module provides the functions needed for querying an XCAP server. |
| **Dialog** | This module is a helper module used by other presence modules to perform some dialog operations. |

The PA module allows the user to use PUBLISH requests to publish presence information. It can handle PUBLISH requests and SUBSCRIBE requests. The PA understands only basic PIDF, but it can handle PIDF extensions such as RPID. Supported document formats in PUBLISH consist of PIDF, CPIM-PIDF, and PIDF extensions (e.g., RPID). Since RPID is a PIDF XML document, it uses the content type application/pidf+xml. We choose to use a PIDF extension as the document format to carry the occupancy information.

In order to identify occupancy information, some new tags have been added for this context information. For example, <area> and <occupancy>. We define a new XML schema in the file "pidf.c". This file is used to parse the tags specified in the PIDF standard. These new tags are created within the <status> element. The <area> tag shows the area name, for example, MINT or OpenArea. The <occupancy> tag shows the occupancy status of this area, for example, Empty, Individual, or Meeting. Additionally, the specific number of persons in the area can be included via the <note> tag. A new schema for occupancy information has been defined and an example of the PIDF in a publish message is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
entity="sip:xueliang@130.237.15.238">
<tuple id="6sJ8J0">
```

```
<status>
        <basic>open</basic>
        <area>MINT</area>
        <occupancy>Empty</occupancy>
</status>
<note> 0 </note>
<contact priority="0.8">ccsmoto</contact>
</tuple>
</presence>
```

The main file within the PA module for handling publishes is "publish.c". The code in this file is responsible for parsing and storing the required fields in the event values. The "handle_publish" function processes the PUBLISH request and generates a response to it. This takes the form of a SIP transaction, comprising a SIP request message followed by one or more SIP response messages. The route logic in the SER configuration file invokes the handle_publish function to handle the PUBLISH method. For details of the modified source code and some other components in PA module refer to Mohammad Z. Eslami's thesis [2].

#### *3.2.3.3 Debugging the SER modules*

During the configuration and adaptation of SER to handle occupancy, there was a need to debug the modified code and the configuration file. There are a number of techniques available to debug SER. According to [56], the main types of SER debugging fall into two categories: (a). Capture the SIP messages and (b). Generate debug information.

We used Wireshark [58] to capture the SIP messages and analyze them. A network interface for communication was chosen and the specific communication protocol or port number used to limit the traffic which was monitored. As a result Wireshark displays all packets sent and received from this specific interface and port.

The second method relies on the debugging support built into SER itself. SER has the ability to generate a vast amount of information that can be used for debugging. The "stderr" file can be redirected to a file to capture this data. Once the data is captured, the level of detail can be varied by increasing (or decreasing) the debug level in the ser.cfg file. A larger value for the debugging level generates more details. Most of the time we used debug level 9.

Finally we can add even more information by inserting into the ser.cfg the command "xlog" at appropriate points where additional debugging information is desired. Please refer to the README file in the xlog module for details.

After the system was implemented, we performed additional system testing and analysis. We utilized the debugging information in this process.

# Chapter 4

# 4 Testing and Analysis

This section describes the meeting detector system level testing and data analysis. The chapter describes a set of tests that were used to validate the meeting detector system and also to obtain feedback to suggest further improvements. Data analysis and system analysis based on the testing process are also described.

## 4.1 Test Methodology

In order to ensure that all the requirements of the system are met, test methodologies are used to guide the whole test process and to define test cases. In our test process, function tests and system integration tests were carried out. The function tests consist of physical sensor tests, tests of the detection in logical sensor, and finally tests of correctly publishing occupancy information to SER. The system integration tests were performed in order to determine the system's flexibility, scalability, and robustness. The accuracy of the occupancy detection will also be measured based on statistical analysis. The tests were organized in the following manner:

| | |
|---|---|
| **Description** | A description section introduces and specifies the particular test. |
| **Purpose** | The purpose section states the objectives of a particular test. |
| **Test Setup** | The test setup section shows an overview of the device setup and the system configuration. |
| **Procedure** | The procedure section describes how to carry out the test and lists all the steps needed to perform a specific test. This is an operational guide as to how to perform a given test. The steps describe activities such as enabling and configuring certain devices or network connections. It also provides references to related setups whenever they are required. |
| **Expected Results** | The expected results section lists the results that the tester should observe while executing a particular test. The expected results could be compared with the actual results later for analysis. |
| **Actual Results** | This section describes the outcome of the specific test. Some analysis could also be included in this section. |

## 4.2 Test Cases

A test case is a sequence of steps to test the correct behavior of functionalities of a system. The purpose of a test case is to describe how you intend to empirically verify that the system being developed conforms to the specifications. In other words, the goal is to show that the system can correctly carry out its intended functions.

A lot of function tests were conducted in conjunction with implementation of the system. Additionally, tests regarding to the beams field of view and waveforms of the HAA52 detector were conducted by Daniel Hübinette and documented in his masters thesis [1]. In this section, we present the meeting detector system testing as a whole.

### 4.2.1 Single Detector Mode

Initially the system utilized a single detector. Thus testing begins with a single physical sensor.

**Description**

The single detector senses an infrared heat source. We examine the raw data from the physical sensor which reveals the waveform of the sensor's output. The collected raw data are sent to the logical sensor through a UDP socket in order to perform the detection. The occupancy information obtained from detection is published to the context server (a SIP proxy) using the SIMPLE protocol.

**Purpose**

Testing of a single detector was performed to validate that the entire meeting detector prototype system works well with a single detector, i.e., that the basic functions operate as designed.

**Test Setup**

The hardware system is configured as Figure 4.1.

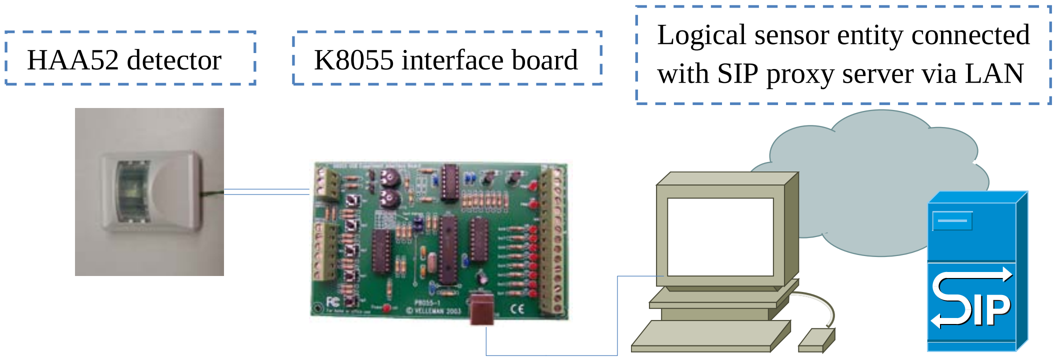

HAA52 detector
K8055 interface board
Logical sensor entity connected with SIP proxy server via LAN
SIP

Figure 4.1 System Setup with Single Detector

The HAA52 detector is powered with an AC adapter and connected to the K8055 interface board with two wires. One is for the analog signal, and the other is a ground wire. The K8055 interface board is also connected to a PC (ccsmoto). The board is powered by this USB connection. This computer acts as both a physical sensor entity and a logical sensor entity. In addition to this, another PC (ccsleft) is configured as a context server (running as a SIP proxy server) and connected with the logical entity via a Local Area Network (LAN) connection (specifically via a 100 Mbps Ethernet connection).

Note that the detector is deployed on a pillar beside the entrance door to a small meeting room (MINT). An optimum placement of the sensor was determined based on earlier tests. The electrical signal changes when a person walks through the door, either in or out.

**Procedure**

1. Make sure the system has been setup as Figure 4.1 and all elements are powered on.
2. Start up the physical sensor program to collect the signal from the USB attached interface board and send it to the logical sensor.
3. Start up the logical sensor program to receive the sensor data and perform the detection calculations.
4. A number of entries and exits are made to gather sensor data. This data was collected based upon one person walking through the door at a normal walking pace.
5. The sensor data is processed in the logical sensor program to determine the number of entries and exits. As a result the number of persons in the room will be calculated.
6. The logical sensor program publishes occupancy information to the context server if the occupancy status changes.

**Expected Results**

The electrical signal is based on a voltage change due to a person walking in or out the door. The entry and exit waveforms can be differentiated. The signal from the physical sensor was correctly digitized by the interface board, set over the successfully collected by the physical entity and sent to logical entity in UDP packets. These packets were monitored by running Wireshark in the system and listening to the loop back interface on the listening port of the logical entity. The logical entity correctly receives the UDP data and processes the sensor data. The occupancy information is correctly calculated by logical program and sent to the SIP proxy server. A 200 OK message is sent by the context server in response to the PUBLISH message. Wireshark also monitors the network traffic send to and received from the context server.

**Actual Results**

The waveforms changes when there is an entry or exit. These signals corresponded to those shown in Figure 3.8. The logical entity receives the data and obtains the occupancy information after processing. The context server also sends a 200 OK message in response to the PUBLISH message from the logical entity.

A detailed data analysis based upon these results can be found in Section 4.3.1.

### 4.2.2 Multiple Detectors Mode

Since the single meeting detector was successfully implemented and verified, we are ready to deploy the meeting detector system in different areas with multiple detectors. This section describes the system testing based on multiple detectors.

**Description**

Multiple detectors can be connected to the same K8055 interface board. We may collect the sensor data from the two detectors simultaneously. The collected raw data are sent to the logical sensor through the same UDP socket program in order to do detection calculation. The physical and logical sensor codes need to be adapted for multiple detectors. Then the occupancy information obtained from detection is published to the context server (a SIP proxy) using the SIMPLE protocol whenever the occupancy status changes (in either of the areas).

**Purpose**

The testing of multiple detectors mode aims to validate that the meeting detector prototype system can be expanded for multiple room, thus it is ready to be deployed in an office environment.

**Test Setup**

The hardware configuration of the system with two detectors is shown in Figure 4.2.

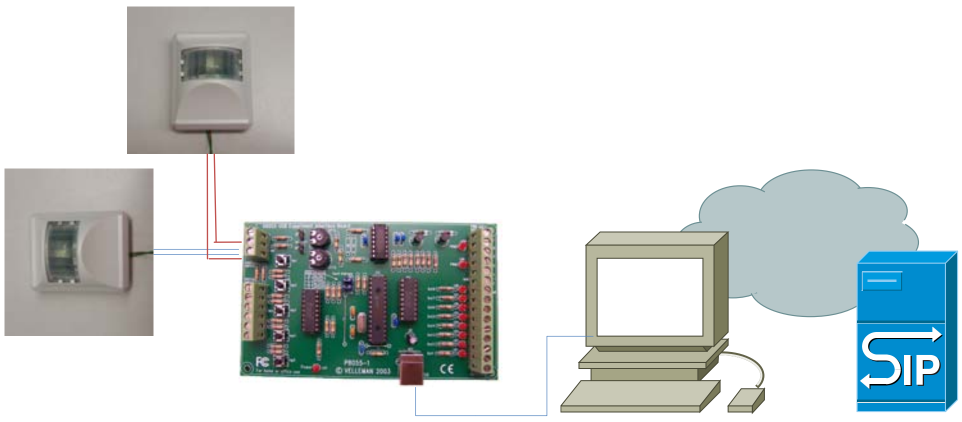


Figure 4.2 System Setup with Two Detectors

**Procedure**

1. Make sure the system has been setup as shown in Figure 4.2 and all elements are powered on.
2. Start up the physical sensor program and the logical sensor program in order.
3. A number of entries and exits are made to gather the sensor data.
   a) Enter each one area and then exit
   b) Enter both areas at the same time and exit respectively
4. The logical program processes the sensor data and obtains the occupancy information based on the entries and exits.
5. Monitor the occupancy information changes and whether it is published to the context server whenever an occupancy status changes.

**Expected Results**

The sensor data from both detectors are successfully collected by the physical entity and sent to logical entity with UDP packets. We monitor the traffic with Wireshark. The logical entity correctly receives the UDP data and processes the sensor data for both detectors respectively. The occupancy information for both areas are correctly calculated by logical program and sent to SIP proxy server. The context server successfully handles the published information.

**Actual Results**

The physical entity can collect the sensor data for multiple detectors simultaneously. The logical entity receives the sensor data and obtains the occupancy information for multiple areas. The logical program makes publish messages to the context server whenever there is an occupancy information changes for either of the areas. The context server also sends a 200 OK message in response to both PUBLISH messages from the logical entity.

A detailed data analysis based upon the results of multiple detectors can be found in Section 4.3.1.

From the iterative testing, we also found that a desirable waveform for detection calculating could be achieved by limiting the viewpoint of PIR detector. We could use two side panels to protect the detector againt abnormal influences since the HAA52 detector had a too wide detection angle of $90^o$ in horizon when someone passes by the detector.

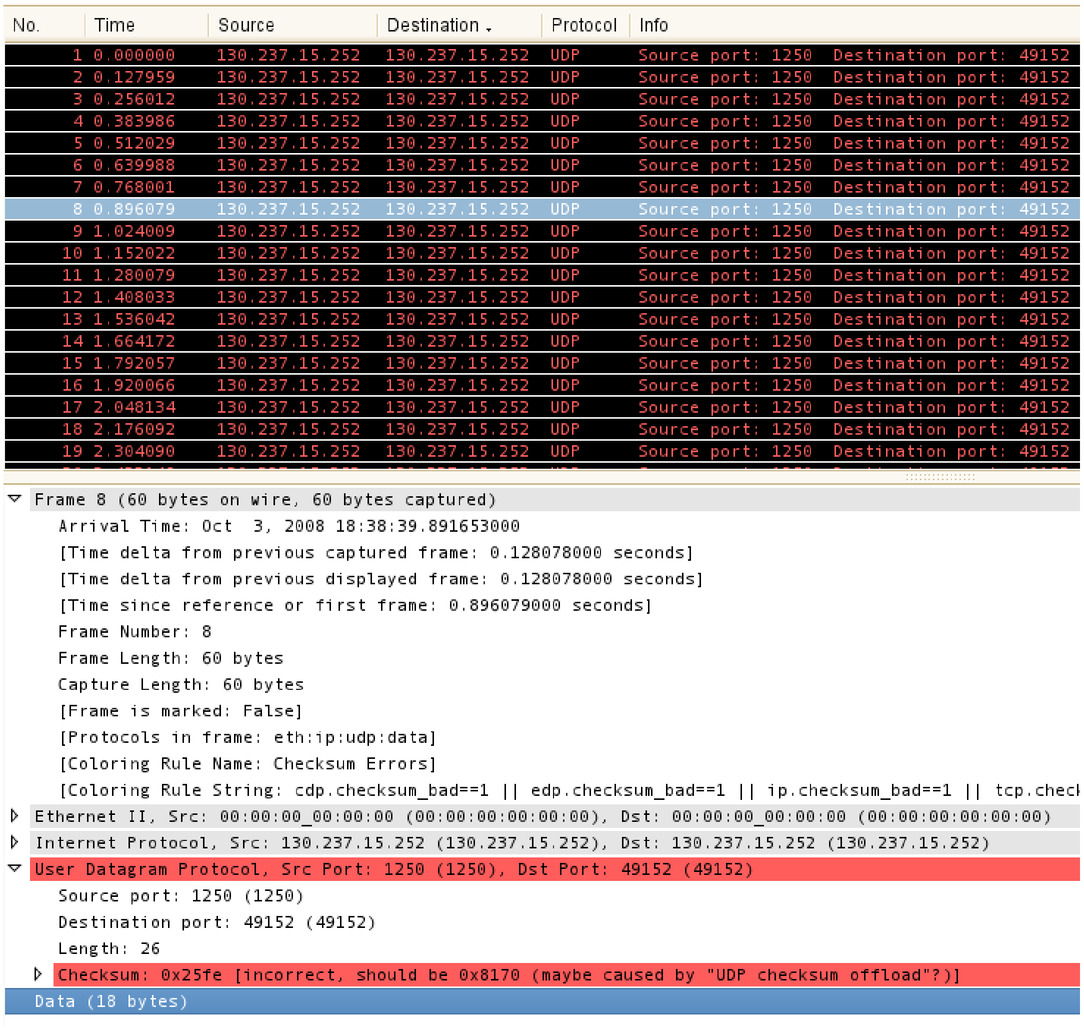
## 4.3 Data Analysis

In order to evaluate and improve the meeting detector system, we perform data and system analysis based on the results of system testing described in Section 4.2.

### 4.3.1 UDP Packets between Physical and Logical Entities

As described in Section 3.2.2.1, UDP packets are used to carry the sensor data from a physical sensor entity to the logical entity. The traffic between the physical and logical sensor entities can be captured using Wireshark for detailed analysis.

An example of collected UDP packets is shown as Figure 4.3.

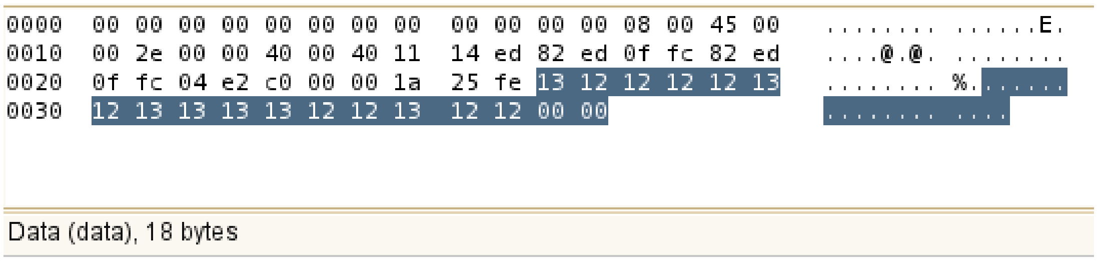


```
No.   Time        Source          Destination     Protocol  Info
    1 0.000000    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
    2 0.127959    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
    3 0.256012    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
    4 0.383986    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
    5 0.512029    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
    6 0.639988    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
    7 0.768001    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
    8 0.896079    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
    9 1.024009    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   10 1.152022    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   11 1.280079    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   12 1.408033    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   13 1.536042    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   14 1.664172    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   15 1.792057    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   16 1.920066    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   17 2.048134    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   18 2.176092    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152
   19 2.304090    130.237.15.252  130.237.15.252  UDP       Source port: 1250  Destination port: 49152

Frame 8 (60 bytes on wire, 60 bytes captured)
    Arrival Time: Oct  3, 2008 18:38:39.891653000
    [Time delta from previous captured frame: 0.128078000 seconds]
    [Time delta from previous displayed frame: 0.128078000 seconds]
    [Time since reference or first frame: 0.896079000 seconds]
    Frame Number: 8
    Frame Length: 60 bytes
    Capture Length: 60 bytes
    [Frame is marked: False]
    [Protocols in frame: eth:ip:udp:data]
    [Coloring Rule Name: Checksum Errors]
    [Coloring Rule String: cdp.checksum_bad==1 || edp.checksum_bad==1 || ip.checksum_bad==1 || tcp.checl
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol, Src: 130.237.15.252 (130.237.15.252), Dst: 130.237.15.252 (130.237.15.252)
User Datagram Protocol, Src Port: 1250 (1250), Dst Port: 49152 (49152)
    Source port: 1250 (1250)
    Destination port: 49152 (49152)
    Length: 26
  Checksum: 0x25fe [incorrect, should be 0x8170 (maybe caused by "UDP checksum offload"?)]
Data (18 bytes)
```

Figure 4.3 UDP Packets from Physical Sensor

As we use the same PC for the physical and logical sensor, the source and destination addresses of the UDP packets are the same IP address (i.e., one of ccsmoto's Ethernet interface IP addresses, in this case the system was configured to use 130.237.15.238 - note that although this interface's address is used, the actual interface which is used by the network stack is the loopback interface - hence the packets are never actually sent via a physical interface, rather there is simply a set of updates to pointers in the networking stack). The destination port is defined as 49152 and the source port is a random port number. Frame 8 is shown in detail in Figure 4.3. The time difference from the previously captured frame is 0.1280078000 seconds (note that not all of these digits are significant).



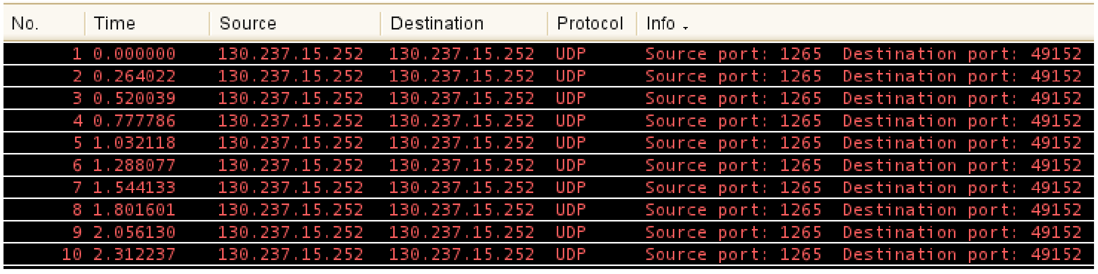
```
0000   00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010   00 2e 00 00 40 00 40 11  14 ed 82 ed 0f fc 82 ed   ....@.@. ........
0020   0f fc 04 e2 c0 00 00 1a  25 fe 13 12 12 12 12 13   ........ %.......
0030   12 13 13 13 13 12 12 13  12 12 00 00               ........ ....

Data (data), 18 bytes
```

Figure 4.4 A Data Frame with a Session Size of 16 bytes.

Figure 4.4 shows a data frame from the physical entity. The digitized sensor readings are highlighted. The offset from the start of the frame is shown in the left hand column in octal; the values of each of the bytes in the frame are shown in the middle two columns, while the right hand column shows the ASCII equivalents of each of the bytes in the frame. The sensor data of this frame is from a single detector with an initial session size of 16 bytes. As we noted in the earlier description of the physical sensor code, we added another two bytes at the end to indicate the sensor's ID (in this case channel 1. which is assigned the ID 0). So the data field is 18 bytes in length. As long as we can know the area where the data is from, additional detectors can be identified using this information.

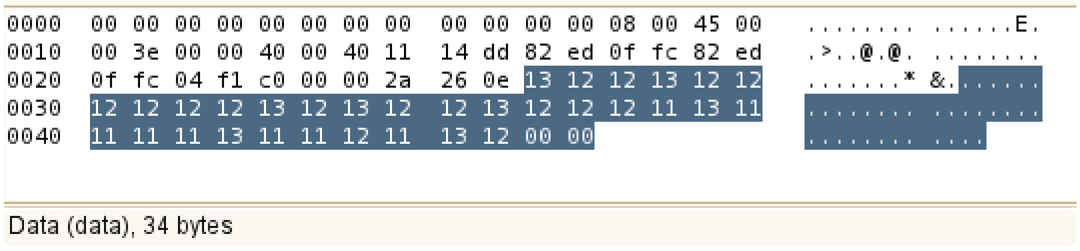
Another example of UDP packets, captured with an initial session size of 32 bytes are shown in Figure 4.5.



```
No.     Time         Source           Destination      Protocol  Info
      1 0.000000     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
      2 0.264022     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
      3 0.520039     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
      4 0.777786     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
      5 1.032118     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
      6 1.288077     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
      7 1.544133     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
      8 1.801601     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
      9 2.056130     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
     10 2.312237     130.237.15.252   130.237.15.252   UDP       Source port: 1265  Destination port: 49152
```

Figure 4.5 UDP Packets with a Session Size of 32 bytes

As we can see from above figure, the interval time between two contiguous captured frames is about 0.256013000 seconds which is approximately twice as long as for packets of 16 bytes. This interframe delay is due to the fact that the physical sensor is being sampled 125 times per second (i.e., every 8 ms) and each sample is digitized as an 8 bit value. This rate is the fastest rate which data can be read from the interface board using the ReadAnalogChannel() call.



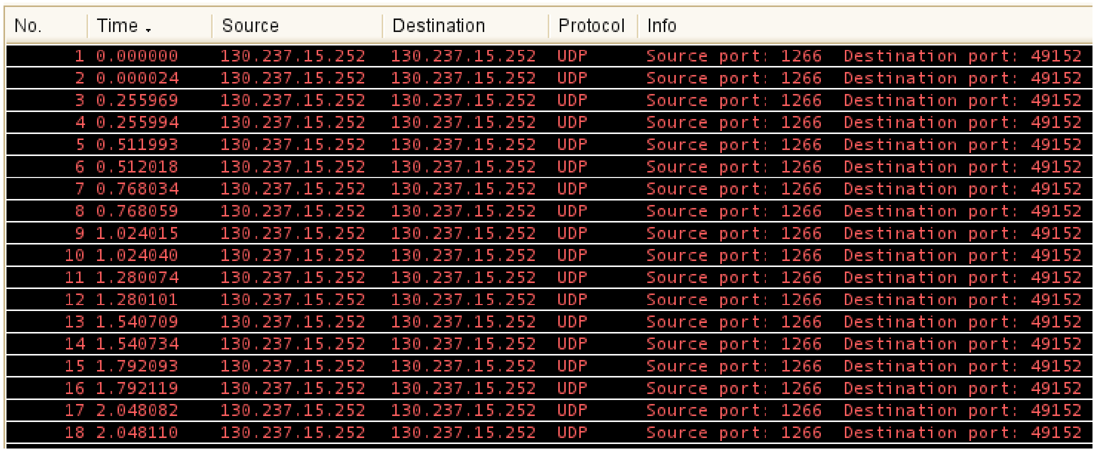
```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 3e 00 00 40 00 40 11  14 dd 82 ed 0f fc 82 ed   .>..@.@. ........
0020  0f fc 04 f1 c0 00 00 2a  26 0e 13 12 12 13 12 12   .......* &.......
0030  12 12 12 12 13 12 13 12  12 13 12 12 12 11 13 11   ........ ........
0040  11 11 11 13 11 11 12 11  13 12 00 00               ........ ....

Data (data), 34 bytes
```

Figure 4.6 A Data Frame with a Session Size of 32 bytes

Figure 4.6 shows a data frame with a session size of 32 bytes. Another two bytes are also added to this frame.

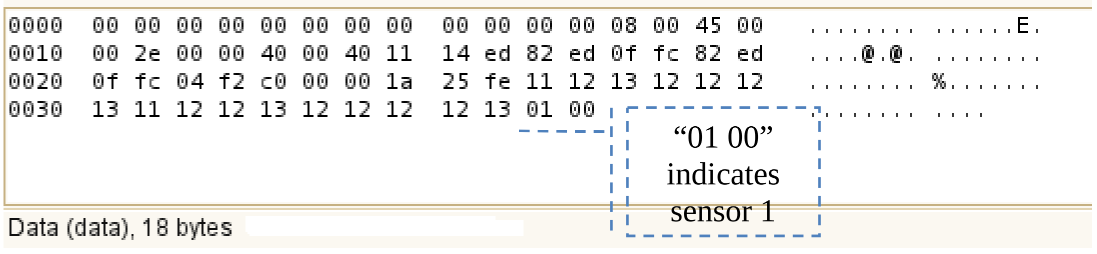
We also performed tests with multiple detectors sending UDP packets simultaneously as shown in Figure 4.7.



```
No.   Time       Source          Destination     Protocol  Info
    1 0.000000   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
    2 0.000024   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
    3 0.255969   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
    4 0.255994   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
    5 0.511993   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
    6 0.512018   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
    7 0.768034   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
    8 0.768059   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
    9 1.024015   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
   10 1.024040   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
   11 1.280074   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
   12 1.280101   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
   13 1.540709   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
   14 1.540734   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
   15 1.792093   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
   16 1.792119   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
   17 2.048082   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
   18 2.048110   130.237.15.252  130.237.15.252  UDP       Source port: 1266  Destination port: 49152
```

Figure 4.7 Received UDP Packets with Multiple Detectors

The UDP packets from multiple detectors are alternating transmission. The interval time is stable. Each packet has an initial session size of 16 bytes and 2 additional bytes for differentiation. A packet from a second detector is shown in Figure 4.8. The last two bytes of "01 00" indicate a sensor id apart from "00 00".



```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  00 2e 00 00 40 00 40 11  14 ed 82 ed 0f fc 82 ed   ....@.@. ........
0020  0f fc 04 f2 c0 00 00 1a  25 fe 11 12 13 12 12 12   ........ %.......
0030  13 11 12 12 13 12 12 12  12 13 01 00               ........ ....

Data (data), 18 bytes
```

Figure 4.8 A Data Frame from a Second Detector

### 4.3.2 Detection Algorithm Analysis

#### *a. State based detection algorithm*

For state based detection, the threshold values are set in the logical sensor code according to the voltage level as shown in Figure 3.7. The threshold variables are: topzone, midtop, midbot, and botzone. These values were defined in advance based on the waveform observed on an oscilloscope. These variables can be adjusted later to increase accuracy. This adjustment was done in conjunction with the testing phase in real environment.

#### *b. Correlation based detection algorithm*

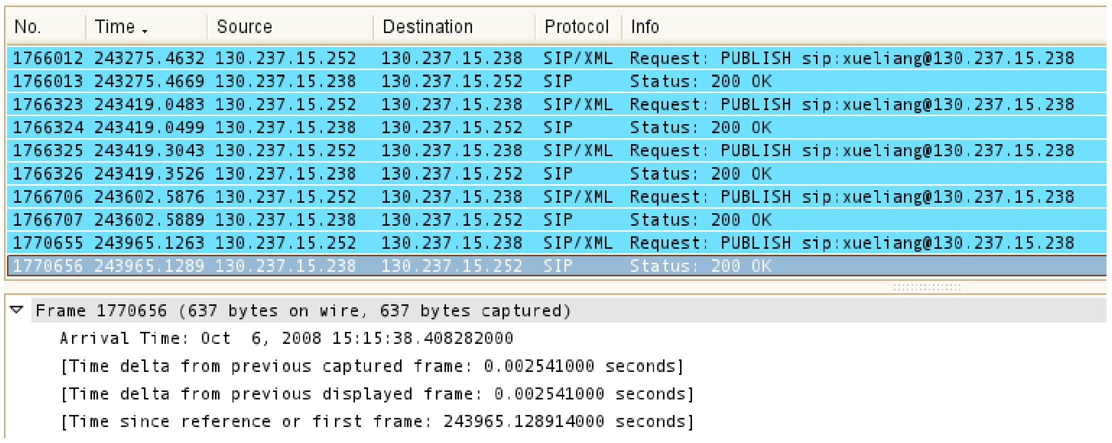
For correlation based detection, a series of correlation template data are collected by having a person walk past the detector in each direction. The correlated data represents a sample waveform. The correlation data both for entry and exit was collected. As described in Section 3.2.2.2, a threshold value for correlation coefficient must be chosen correctly in order to achieve good accuracy. However, the sensor system is not 100% accurate in detecting the actual entries and exits.

While using measurements to determine a reasonable threshold value, we found that correlating the entry and exit data against a single correlation template data as Daniel Hübinette did in his thesis project did not produce the expected high correlation that we desired. So we chose two series of correlation template data, one for entry, and the other for exit. Both of them were used to search for a positive correlation. For double detectors, we did the same thing. Thus two series of data are used to calculate the correlation coefficient for each detector. The correlation based detection algorithm in the logical sensor code is capable of dealing with two detectors and can easy to be adapted to work with more than two detectors. Note that the two different HAA52 detectors were found to differ somewhat, but no systematic study was performed to understand the variance between HAA52 detectors. However, such a study should be carried out if a mass produced integrated detector was to be produced to understand if it is possible to avoid having to collect training data for each individual detector. It might also be possible to accommodate the difference between individual detectors by post processing the data collected from each detector - however as this was not the focus of this work, this was not done either.

Another issue to be noted is that we programmed the logical sensor code to send PUBLISH messages to the context server whenever there was a change of occupancy information. So the number of persons in an area could be monitored by the context server in (near) real-time. The overall delay between a person passing the IR detector and the SIP Publish message being set is described in section 4.3.3.

### 4.3.3 SIP messages between Logical Entity and Context Server

The SIMPLE protocol is used to distribute context information to a context server. In the meeting detector system, the logical sensor entity sends SIP PUBLISH messages to the SER server. The SER server replies a 200 OK message in response to a successful PUBLISH message. A series of SIP messages captured by Wireshark are shown in Figure 4.9.



```
No.     Time        Source          Destination     Protocol Info
1766012 243275.4632 130.237.15.252  130.237.15.238  SIP/XML  Request: PUBLISH sip:xueliang@130.237.15.238
1766013 243275.4669 130.237.15.238  130.237.15.252  SIP      Status: 200 OK
1766323 243419.0483 130.237.15.252  130.237.15.238  SIP/XML  Request: PUBLISH sip:xueliang@130.237.15.238
1766324 243419.0499 130.237.15.238  130.237.15.252  SIP      Status: 200 OK
1766325 243419.3043 130.237.15.252  130.237.15.238  SIP/XML  Request: PUBLISH sip:xueliang@130.237.15.238
1766326 243419.3526 130.237.15.238  130.237.15.252  SIP      Status: 200 OK
1766706 243602.5876 130.237.15.252  130.237.15.238  SIP/XML  Request: PUBLISH sip:xueliang@130.237.15.238
1766707 243602.5889 130.237.15.238  130.237.15.252  SIP      Status: 200 OK
1770655 243965.1263 130.237.15.252  130.237.15.238  SIP/XML  Request: PUBLISH sip:xueliang@130.237.15.238
1770656 243965.1289 130.237.15.238  130.237.15.252  SIP      Status: 200 OK

▽ Frame 1770656 (637 bytes on wire, 637 bytes captured)
    Arrival Time: Oct  6, 2008 15:15:38.408282000
    [Time delta from previous captured frame: 0.002541000 seconds]
    [Time delta from previous displayed frame: 0.002541000 seconds]
    [Time since reference or first frame: 243965.128914000 seconds]
```

Figure 4.9 SIP PUBLISH Messages

As we can see, the PUBLISH message has a source address of 130.237.15.252 and a destination address of 130.237.15.238 which means it is sent from logical entity (ccsmoto) to the context server (ccsleft). A captured PUBLISH message is shown below:

```
Session Initiation Protocol
    Request-Line: PUBLISH sip:xueliang@130.237.15.238 SIP/2.0
        Method: PUBLISH
    Message Header
        Via: SIP/2.0/UDP 130.237.15.252:5060;branch=z9hG4bK6sJ8J0y
            Transport: UDP
            Sent-by Address: 130.237.15.252
            Sent-by port: 5060
            Branch: z9hG4bK6sJ8J0y
        To: <sip:ccsleft@130.237.15.238>
            SIP to address: sip:ccsleft@130.237.15.238
        From: <sip:ccsmoto@130.237.15.252>;tag=6sJ8
            SIP from address: sip:ccsmoto@130.237.15.252
            SIP tag: 6sJ8
        Call-ID: 288@130.237.15.252
        CSeq: 1 PUBLISH
        Max-Forwards: 10
        Expires: 5
        Event: occupancy
```

```
Content-Type: application/pidf+xml
Content-Length: 358
```

The occupancy information is carried in the message body in PIDF format. The corresponding content is shown below:

```
Message body
    eXtensible Markup Language
        <?xml version="1.0" encoding="UTF-8" ?>
        <presence
            xmlns="urn:ietf:params:xml:ns:pidf"
            entity="sip:ccsleft@130.237.15.238">
            <tuple id="6sJ8J0">
                <status>
                    <basic> open </basic>
                    <area> MINT </area>
                    <occupancy> Individual </occupancy>
                    </status>
                <note> 1 </note>
                <contact priority="0.8"> ccsmoto </contact>
                </tuple>
            </presence>
```

The occupancy information is contained within the <status> tag. The message above shows that the room MINT is occupied by 1 person. The PIDF extension format was described in Section 3.2.3.2.

In response to the PUBLISH message, a 200 OK message is sent by the SER server, this message was also captured by Wireshark. We notice that there is a delay between the PUBLISH message and 200 OK message. Ten groups of PUBLISH & 200 OK messages were chosen from the data collected Wireshark and the delay between the PUBLISH and the corresponding 200 OK has a mean value of 0.002378 seconds. This represents the mean time required by the context server to handle the PUBLISH message and the 200 OK response message. Note that these measurements were made when the computers involved were only doing typical background processing of other tasks; thus these values represent lower bounds on the expected response time - which may not be representative of the delay for a highly loaded context server. Apart from the processing time for context server, the detection mechanism also takes couples of seconds to perform correlation based detection since the number of correlation samples needs to correlate the waveform change when a person passes by. So the main delay of the system is caused by the sample rate and correlation, basically the waveform itself. In comparison with the detection delay, the delay on context server can be ignored.

## 4.4 Evaluation

This section describes the evaluation of the meeting detector system. We will evaluate and determine the accuracy and efficiency of the system. Some discussion about achievements and improvements with regard to the original goals are included in this section.

### 4.4.1 System Evaluation

System evaluation is based on some evaluation criteria. For this meeting detector system, we mainly focus in the evaluation on accuracy, robustness, and scalability of the system.

#### *4.4.1.1 Accuracy*

At the beginning of this thesis project, we intended to implement the system and deploy it in an office environment on a large scale. In order to justify this minimum accuracy must be achieved.

A statistical model can be established to determine the accuracy of the detection. We define the conditions: an area is occupied (with the correct number of persons in the area) or empty; the occupied status is detected or not. There are four outcomes: True Positive, False Positive, False Negative, and True Negative. From Table 4.1 we can see that the desired values correspond to the correct results: True Positive and True negative.

Table 4.1 Accuracy Determination

| | Detected | Not Detected |
|---|---|---|
| Occupied | True Positive | False Negative |
| Empty | False Positive | True Negative |

The test results can be sorted using this model to calculate the accuracy of the system. From our analysis, we find the state based detection is reliable with a simple fixed waveform, such as a sine wave or cosine wave, but does not work well for this application. However, the correlation based detection is reliable with most of waveforms; since it does not care about classifying the measured signal voltage into one of a small range of values. In the correlation based detection, we easily use multiple templates to detect the entry and exit separately. While the statistical accuracy increases with the correlation based detection method, so does the CPU load. However, the effort required to compute the correlation is not significant, as the number of processor cycles and the frequency with which it must compute the correlation leads to a rather small load as compared to other applications running on the PC.

Note that the detection for the detector deployed for the OpenArea does not perform as well as the one in the room MINT since the waveform from the detection of persons entering and exiting this open environment is sometimes abnormal. The accuracy for the small conference room (MINT) detector is shown in Table 4.2 and for the OpenArea in Table 4.3.

Table 4.2 Detection Accuracy in MINT

| Threshold | True | False | Accuracy |
|---|---|---|---|
| 0.60 | 10 | 10 | 0.50 |
| 0.65 | 9 | 11 | 0.45 |
| 0.70 | 11 | 9 | 0.55 |
| 0.75 | 14 | 6 | 0.70 |
| 0.80 | 15 | 5 | 0.75 |
| 0.85 | 15 | 5 | 0.75 |
| 0.90 | 11 | 9 | 0.55 |
| 0.95 | 8 | 12 | 0.40 |
| 1.00 | 0 | 20 | 0 |

Table 4.3 Detection Accuracy in OpenArea

| Threshold | True | False | Accuracy |
|---|---|---|---|
| 0.60 | 9 | 11 | 0.45 |
| 0.65 | 9 | 11 | 0.45 |
| 0.70 | 11 | 9 | 0.55 |
| 0.75 | 10 | 10 | 0.50 |
| 0.80 | 12 | 8 | 0.60 |
| 0.85 | 12 | 8 | 0.60 |
| 0.90 | 8 | 12 | 0.40 |
| 0.95 | 5 | 15 | 0.25 |
| 1.00 | 0 | 20 | 0 |

According to the accuracy testing results, we can see that the detector in the meeting room MINT has a better accuracy than the one for OpenArea. Apart from this, the better accuracy comes with the threshold around 0.80 for correlation detection method.

#### *4.4.1.2 Robustness*

The system seems to be very stable. The HAA52 PIR detector with the K8055 interface board worked well for more than one week without a break. The physical and logical sensor program was running on the openSUSE Linux and calculating the occupancy status correctly. However, the accuracy over a long period of operation can not be guaranteed because of some anomaly in the waveform always happens in the lab environment. This anomaly occurs very infrequently and we have not been able to understand exactly what causes it - as it occurs too infrequently. For this reason we reset the room occupancy for each room to zero late at night (after there has been no change in room occupancy for 5 hours).

#### *4.4.1.3 Scalability*

The meeting detector system can be easily extended with multiple detectors. Multiple Velleman K8055 interface boards can be connected to a USB hub. Using a USB connection to the computer has both advantages and disadvantages. On one hand, the USB cable can power supply the board and provides high speed communication. On the other hand, it is not practical to deploy long USB cables, as using a long cable required installing USB hub (hence limiting the practical distance between the computer and the Velleman K8055 board to ~5 meters. However, the cable between the interface board and the HAA52 sensor can be several meters long. Due to practical limitation, we have not measured the signal loss over long cables. This remains for future testing.

The physical sensor program can read signals from multiple detectors with a simple modification to the program. The logical sensor code can also be programmed to handle multiple detectors and interface boards (note that as this program receives packets, the physical sensor nodes could be located elsewhere on the network). Sensor data can be simultaneously calculated to obtain the occupancy information. The SIP proxy server also receives the PUBLISH messages via the network, so the physical location of the physical sensor is not an issue. A practical limitation is that the k8055 library only supports 4 boards (the number of boards which can have unique ID numbers), thus connecting more than 8 sensors to a single computer would require modifications to this library to enable it to use the USB bus addresses to distinguish more than 4 K8055 boards. This approach is probably leading in the wrong direction, since as noted earlier it would be better to make an integrated HAA52 with perhaps some local signal processing as an Ethernet power and connected device. This would facilitate installation of the devices at distance of up ~100m from the power over Ethernet switch. Using a network attached physical sensor node would require that there be some configuration file to map the MAC address/IP address of each device to a room ID. Potentially this could either be done by a suitable naming convention, a configuration file, or with DNS resource records. However, the design and evaluation of such a system remains for future work. While the design of the system scales well, the use of the Velleman K8055 is not a scalable solution, given the cost of each interface board (599 SEK pre-built or 399 SEK as a kit) versus the 279 SEK for the HAA52 IR motion detector plus ~140 SEK for a AC to DC adaptor to power the HAA52.

### 4.4.2 Achievement of the Goals

As described in Section 1.2, our aim was to design, develop, evaluate, and improve a meeting detector system based on occupancy sensing. Currently, a working prototype is deployed in a lab environment at Wireless@KTH. The meeting detector system successfully collects sensor data from the area, uses this data to perform detection, and provides the context information to a SIP proxy server. We have also completed the testing and basic data analysis.

Based on this, we can see some improvements compared with the previous project by Daniel Hübinette. First, I have designed the meeting detector system with a simple and effective hardware setup. Second, I have developed the detection program to handle multiple detectors. The accuracy for the detector algorithm has also been improved by using two series of correlation templates data. Third, Daniel did the basic sensor detection, but he did not make publish messages actually. I have followed up to explore the publish messages with a SIP proxy server. The context information could be published to a context server which has been implemented to handle the publish messages (occupancy information). The logical sensor entity has also been improved to make a publish message whenever there is an occupancy status change – it means whenever someone passes by the detector. Fourth, I have deployed the meeting detector system in a lab environment. The prototype seems to be a ready-to-use system, although some improvements may be needed before deploying it for a more extensive field trial. Therefore we have achieved most of the goals of this thesis project and we are going to conclude it in the next chapter.

# Chapter 5

# 5 Conclusions and Future Work

In this chapter, we would conclude this masters thesis and propose some suggestions for the enhancement of this system along with other suggestions for future work.

## 5.1 Conclusions

In this thesis, we have successfully designed, implemented, deployed, and evaluated a prototype of a meeting detector system to provide additional context information to a SIP proxy server (SER). Through the literature study of existing sensing technology, we realized that it was possible to design a meeting detector system that could monitor the boundaries of an area to determine the occupancy of this area. This context information could be used by a variety of context-aware systems. A prototype of the meeting detector system was designed and implemented using a Velleman HAA52 PIR detector, a Velleman K8055 interface board, and a PC to create both a physical sensor entity and a logical entity. Another PC running SER was used as a context server.

During this thesis project, we have also tested the system and evaluated this prototype in an experimental setup. First the detection function in the system was tested and shown to operate correctly. Then a series of tests were conducted to guide improvements. These tests were based on data collected in a lab environment. The accuracy of two different detection algorithms was measured and some improvements were suggested based upon the analysis phase. We concluded that using a correlation based algorithm gave the greater accuracy and that separate templates should be used for recognizing a user entering and a user leaving the monitored area. However, some unfinished analysis remains for future work (specifically to isolate the cause for the rare anomalies in detection and to evaluate the long term (longer than 1 week) operating accuracy of the system).

For context information distribution, we used a SER with its presence module as a SIP proxy server. The occupancy information obtained from detection process was sent to the SIP proxy server using SIP/SIMPLE protocol in PIDF format and the context information was stored in a MySQL database in this context server. No significant bottlenecks were observed in this system, which is not surprising given the very low rate of changes in a room's occupancy. While no systemic study has been made with regard to what the actual scaling of this part of the system is, the earlier work by Mohammad Z. Eslami suggests that this context processing is highly scalable.

Evaluation of the system revealed that the meeting detector system had reasonable accuracy, high scalability, and sufficient robustness to operate continuously for one week. Thus the objectives of this thesis project were mostly achieved. However, in order to fulfill the specific requirements for the proposed architecture in Figure 1.1, we would like to improve the accuracy further and make use of the occupancy information as part of a complete context-aware system.

In the next section, some suggestions for future work are given - should anyone wish to continue this project.

## 5.2 Future Work

Several suggestions with regard to future work are described in this section. The emphasis is on near-term activities, but with hints for longer term efforts.

### 5.2.1 Accuracy Improvement in a Live Open Environment (Rather than a Lab Environment)

The most important aspect of the meeting detector system is its accuracy. While the system was reliable and accurate in a controlled laboratory environment, it did not perform very well in a live open environment. Therefore, the detection algorithm should be optimized to better handle the greater variance in the signal. Moreover optimum placement of the detector should also be determined by repeated testing and analysis. Additionally, multiple detectors might be used to monitor the same area in order to improve the accuracy. For example, two sequential sensors could be used to perform detection if more persons pass through the area boundaries at the same time. As suggested in section 5.2.5 perhaps those motion detectors could be augmented by additional sensors.

### 5.2.2 Sensor Software Development

As we have integrated the physical and logical entities within the same computer running Linux, the two programs might be merged into a single program and ported to a low cost embedded environment (such as described in section 5.2.5) which could save the cost for extension use.

Furthermore, the PUBLISH refresh, modify, and remove messages could be implemented to handle information exchange between context entities. For example, the logical sensor entity should send a PUBLISH modify message to change the occupancy information on SIP proxy server if it observes that the last publish message was wrong. The retransmission of the publish messages should also be considered if there is no 200 OK message within a defined time period.

While we have used command line interfaces for the application, a version without any user interface should be developed if the devices are to be replicated and installed in many locations. In conjunction with this an analysis needs to be made of judging how much administrative time is needed to incorporate each sensor node into a large system (for example, for a deployment to monitor the 5 conference rooms and meeting areas in the department or the much large number of such rooms and project rooms in an academic building).

### 5.2.3 Extension to Multiple Areas

In our work two detectors for two areas have been deployed in a lab environment. More detectors could be deployed using additional interface boards connected to a USB hub. But there are limitations on this interface board (four boards at most, two channels for each one). As described in section 4.4.1.3, using such USB attached boards does not seem to scale well for large use. We intend to consider some more integrated physical sensors as described in section 5.2.5.

### 5.2.4 Security Mechanism

Some steps to implement a security mechanism for the communication between context distribution entities may be needed for commercial deployment. For example, an authentication between a logical sensor entity and a SIP proxy server is needed. In addition, it will be important to protect the data coming from the physical sensor to the logical sensor. This might utilize SRTP or some other mechanism. Additionally, a suitable key exchange mechanism needs to be utilized. One simple method might be to include a key at the time of manufacturing of the device, either in the form of a public/private key pair or by a fixed key which could be printed on the back of the device or included along with the device when it is manufactured. There are quite a number of issues regarding how to cost effectively enter these keys into a more complete occupancy detector system which was deployed on a large scale.

### 5.2.5 A More Integrated Physical Sensor

As noted in section 4.4.1.3 a highly integrated IR motion detector and physical sensor that would be Ethernet power and connected could reduce the cost of monitoring multiple rooms. This is an obvious area for future development and is probably the next required step in the evolution of a room occupancy system. An obvious question which arises is what other sensors should be integrated with this, to enable environmental monitoring for a low incremental costs. For example, it would be possible to measure the room temperature, the amount of oxygen and $CO_2$, humidity, etc. Some of these measurements could be provided to a building management system to provide better (and more energy efficient) building operations. Additionally, some of these sensors' values could be used to improve the accuracy of the room occupancy system. There are some other uses for the sensor data including safety and security applications. Anyway, it clearly offers a rich set of issues for a future project to explore.

## References

[1] Daniel Hübinette, “Occupancy Sensor System: For Context-aware Computing”, Master thesis, Communication Systems, Royal Institute of Technology (KTH), Stockholm, Sweden, December 2007

[2] Mohammad Z. Eslami, “A Presence server for Context-aware applications”, Master thesis, Communication Systems, Royal Institute of Technology (KTH), Stockholm, Sweden, December 2007

[3] Yu Sun, “Context-aware applications for a Pocket PC”, Master thesis, Communication Systems, Royal Institute of Technology (KTH), Stockholm, Sweden, December 2007

[4] Bill N. Schilit, Norman I. Adams, and Roy Want, “Context-Aware Computing Applications”, In Proceedings of the Workshop on Mobile Computing Systems and Applications, pages 85-90, Santa Cruz, CA, December 1994

[5] A. K. Dey and G. D. Abowd, “Towards a Better Understanding of Context and Context-Awareness”, In the Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI 2000), April 2000

[6] A. K. Dey, “Providing Architectural Support for Building Context-Aware Applications”, Ph. D. Dissertation, College of Computing, Georgia Institute of Technology, December 2000

[7] Frazer Bennett, Tristan Richardson, and Andy Harter, “Teleporting - Making Applications Mobile”, In Proceedings of 1994 Workshop on Mobile Computing Systems and Applications, pages 82-84, Santa Cruz, December 1994

[8] Abhaya Asthana, Mark Cravatts, and Paul Krzyzanowski, “An indoor wireless system for personalized shopping assistance”, In IEEE Workshop on Mobile Computing Systems and Applications, pages 69-74, Santa Cruz, December 1994

[9] Lidan Hu, “Personal Video: An Intelligent Presentation System”, Master thesis, Communication Systems, Royal Institute of Technology (KTH), Stockholm, Sweden, December 2008

[10] Harry Chen, “An Intelligent Broker Architecture for Context-Aware Systems”, Ph.D. Dissertation, University of Maryland, Baltimore County, December 2004

[11] Ken Hinckley et al., “Sensing Techniques for Mobile Interaction”, Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology, pages 91-100, San Diego, California, United States, 2000

[12] The Context Toolkit, available at: http://www.cs.cmu.edu/~anind/context.html, last visited March 2008

[13] Haruumi Shiode, “In-building Location Sensing Based on WLAN Signal Strength”, Master thesis, Communication Systems, Royal Institute of Technology (KTH), Stockholm, Sweden, March 2008.

[14] Matthias Baldauf and Schahram Dustdar, “A survey on context-aware systems”, International Journal of Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, pages 263-277, June 2007

[15] AXIS NetEye 200+ Support, available at: http://www.axis.com/techsup/cam_servers/cam_200p/s, last visited April 2008

[16] XML, available at: http://en.wikipedia.org/wiki/XML, last visited March 2008

[17] SGML, available at: http://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language, last visited March 2008

[18] XML Tutorial in W3C, available at: http://www.w3schools.com/xml/default.asp, last visited March2008

[19] T. Strang and C. Linnhoff-Popien, “A context modeling survey,” First International Workshop on Advanced Context Modelling, Reasoning and Management as Part of UbiComp 2004, the 6th International Conference on Ubiquitous Computing, pages 33-40, September 2004

[20] Session Initiation Protocol (SIP), available at: http://en.wikipedia.org/wiki/Session_Initiation_Protocol, last visited March 2008

[21] J. Rosenberg et al., “SIP: Session Initiation Protocol,” RFC 3261, IETF, June 2002, available at: http://www.ietf.org/rfc/rfc3261.txt, last visited March 2008

[22] Ubiquity Software Corporation Plc (now part of Avaya), “Understanding SIP - Today's Hottest Communications Protocol”, White Paper, September 2004, available at: http://www.sipcenter.com/sip.nsf/html/WEBB5YNVK8/$FILE/Ubiquity_SIP_Overview.pdf, last visited March 2008

[23] Henry Sinnreich and Alan B. Johnston, “Internet Communications Using SIP: Delivering VoIP and Multimedia Services with Session Initiation Protocol”, 2nd Edition, Wiley, August 2006, ISBN: 0-471-77657-2

[24] ChenXin Zhang, “SIP and Application Internetworking”, Telecommunication Software and Multimedia Lab, Helsinki University of Technology, spring 2003

[25] Session Initiation Protocol Gateway Call Flows and Compliance Information, Cisco Carrier Sensitive Routing User Guide, Cisco Systems Inc., available at: http://www.cisco.com/application/pdf/en/us/guest/products/ps4032/c2001/ccmigration_09186a00800c4bb1.pdf, last visited March 2008

[26] SER, iptel, available at: http://www.iptel.org, last visited March 2008

[27] Jiri Kuthan, “SIP and SER: More Than You Ever Wanted to Know About”, September 2003, available at: http://voip.internet2.edu/meetings/slides/200310/SIP_Express_Router.pdf, last visited March 2008

[28] SIP for Instant Messaging and Presence Leveraging Extensions, available at: http://www.ietf.org/html.charters/simple-charter.html, last visited March 2008

[29] M. Day, J. Rosenberg, and H. Sugano, “A Model for Presence and Instant Messaging”, RFC 2778, IETF, February 2000, available at: http://www.ietf.org/rfc/rfc2778.txt , last visited March 2008

[30] H. Sugano et al., “Presence Information Data Format (PIDF)”, RFC 3863,, IETF, August 2004, available at: http://www.ietf.org/rfc/rfc3863.txt, last visited March 2008

[31] J. Peterson, “Common Profile for Presence (CPP)”, RFC 3859, IETF, August 2004, available at: http://www.rfc-editor.org/rfc/rfc3859.txt, last visited March 2008

[32] J. Rosenberg, “A Data Model for Presence”, RFC 4479, IETF, July 2006, available at: http://www.ietf.org/rfc/rfc4479.txt, last visited March 2008

[33] H. Schulzrinne. et al., “RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)”, RFC 4480, IETF, July 2006, available at: http://www.ietf.org/rfc/rfc4480.txt, last visited March 2008

[34] H. Schulzrinne, “Timed Presence Extensions to the Presence Information Data Format (PIDF) to Indicate Status Information for Past and Future Time Intervals”, RFC 4481, IETF, July 2006, available at: http://www.ietf.org/rfc/rfc4481.txt, last visited March 2008

[35] H. Schulzrinne, “CIPID: Contact Information for the Presence Information Data Format”, RFC 4482, IETF, July 2006, available at: http://www.ietf.org/rfc/rfc4482.txt, last visited March 2008

[36] J.Lennox, X.Wu, and H.Schulzrinne. "CPL: A Language for User Control of Internet Telephony Services", RFC 3880, IETF, October 2004, available at: http://www.ietf.org/rfc/rfc3880.txt, last visited March 2008

[37] Alisa Devlic, "Extending CPL with context ontology", In Mobile Human Computer Interaction (Mobile HCI 2006) Conference Workshop on Innovative Mobile Applications of Context (IMAC), Espoo, Finland, September 2006

[38] W. Steven Conner, John Heidemann, Lakshman Krishnamurthy, Xi Wang, and Mark Yarvis, "Workplace Applications of Sensor Networks", Technical Report ISI-TR-2004-591, USC/Information Sciences Institute, July, 2004. To appear as a chapter in Wireless Sensor Networks: A Systems Perspective, Nirupama Bulusu and Sanjay Jha, editors; Artech House, publisher (2004)

[39] MICA2 Mote, a third generation mote module, available at: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf, last visited March 2008

[40] Chris Chan and Michael Onorato, "Room Occupancy Detection with Power Line Positioning in a Wireless Sensor Network", Final project paper in COS479 Pervasive Information Systems, Princeton University, May 2007

[41] Shwetak N. Patel, Khai N. Truong, and Gregory D. Abowd, "Power Line Positioning: A Practical Sub-Room-Level Indoor Location System for Domestic Use", In the proceedings of Ubicomp 2006, pages 441-458, September 2006

[42] Panasonic ZigBee wireless communication module, available at: http://www.panasonic.com/industrial/components/modules/mod_rfm.html, last visited March 2008

[43] Yoosoo Oh, Albrecht Schmidt, and Woontack Woo, "Designing, Developing, and Evaluating Context-Aware Systems", 2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07), pages 1158-1163, April 2007

[44] Jue Wang, Guanling Chen, and David Kotz, "A Sensor-fusion Approach for Meeting Detection", In Workshop on Context Awareness at the Second International Conference on Mobile Systems, Applications, and Services (MobiSys 2004), Boston, MA, June 2004

[45] Shameem Ahmed, Moushumi Sharmin, and Sheikh I. Ahamed, "A Smart Meeting Room with Pervasive Computing Technologies", Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks, pages 366-371, May 2005

[46] PIR Intrusion Detector, HA552, Velleman Components N.V., available at: http://www.vellemanusa.com/us/enu/product/view/?id=351031, last visited September 2008

[47] USB Experiment Interface Board, K8055, Velleman Components N.V., available at: http://www.vellemanusa.com/us/enu/product/view/?id=500349, last visited September 2008

[48] Illustrated Assembly Manual, Velleman Components N.V., available at: http://www.vellemanusa.com/downloads/0/illustrated/illustrated_assembly_manual_k8055_uk_rev3.pdf, last visited September 2008

[49] Mark T. Smith and Gerald Q. Maguire Jr., SmartBadge/BadgePad version 4, HP Labs and Royal Institute of Technology (KTH), available at: http://www.it.kth.se/~maguire/badge4.html, last visited September 2008

[50] Linux K8055 library for Velleman USB interface board, Sven Lindberg, available at: http://prdownloads.sourceforge.net/libk8055/libk8055.0.2.tar.gz, last visited September 2008

[51] Linux Graphical Interface for K8055 interface board, available at: http://vellemank8055.googlepages.com/, last visited September 2008

[52] User Datagram Protocol, available at: http://en.wikipedia.org/wiki/User_Datagram_Protocol, last visited September 2008

[53] Correlation, available at: http://en.wikipedia.org/wiki/Correlation, last visited September 2008

[54] SIP Express Router Source Code version 2.0, iptel.org, available at: http://ftp.iptel.org/pub/ser/2.0.0/src/ser-2.0.0_src.tar.gz, last visited September 2008

[55] SERCTL, iptel.org, available at: http://ftp.iptel.org/pub/serctl/, last visited September 2008

[56] Paul Hazlett, Simon Miles, and Greger V. Teigre, “SER - Getting Started”, available at: http://siprouter.teigre.com/doc/gettingstarted/ch04.html, last visited September 2008

[57] Vaclav Kubart, “SER presence handbook”, available at: http://www.iptel.org/~vku/presence_handbook/, last visited September 2008

[58] Wireshark, a network protocol analyzer, available at: http://www.wireshark.org/, last visited September 2008

## Appendix A: K8055 Test Program

```
/*
This code was modified by G. Q. Maguire Jr. and Xueliang Ren to collect some
analog data from channel 1 and write it to a file.

//Command to compile this test program
gcc -o readtest -lusb -L/usr/lib -lm -lk8055 readtest.c
*/

#include <string.h>
#include <stdio.h>
#include <usb.h>
#include <assert.h>
#include <sys/time.h>
#include "k8055.h"

/* The IP address and port number for the sensor system */
#include "address.h"

#define STR_BUFF 256
#define false 0
#define true 1

extern int DEBUG;

int ia1 = -1;
int ia2 = -1;
int id8 = -1;
int ipid = 0;

int numread = 1;

int debug = 0;

int dbt1 = -1; // (-1 => not to set)
int dbt2 = -1; // (-1 => not to set)

int resetcnt1 = false;

int delay = 0;

/*
        Convert a string on n chars to an integer
        Return    1 on sucess
                        0 on failure (non number)
*/
```

```
int Convert_StringToInt(char *text, int *i)
{
        return sscanf(text, "%d", i);
}

/*
        Write help to standart output
*/
void display_help ( char *params[] ) {

        printf("K8055 version 0.4 MrBrain Build\n");
        printf("Copyright (C) 2004 by Nicolas Sutre\n");
        printf("Copyright (C) 2005 by Bob Dempsey\n");
        printf("Copyright (C) 2005 by Julien Etelain and Edward Nys\n");
        printf("Copyleft (L) 2005 by Sven Lindberg\n");
        printf("\n");
        printf("Syntax : %s [-p:(number)] [-d:(value)] [-a1:(value)] [-
a2:(value)]\n",params[0]);
       printf("                [-num:(number)  [-delay:(number)]  [-dbt1:(value)]\n");
        printf("                [-dbt2:(value)] [-reset1] [-debug]\n");
        printf("        -p:(number)    Set board number\n");
        printf("        -a1:(value)     Set analog output 1 value (0-255)\n");
        printf("        -a2:(value)     Set analog output 2 value (0-255)\n");
        printf("        -num:(number)      Set number of measures\n");
        printf("        -delay:(number) Set delay between two measure (in msec)\n");

        printf("        -dbt1:(value)      Set debounce time for counter 1 (in msec)\n");

        printf("        -dbt2:(value)      Set debounce time for counter 2 (in msec)\n");

        printf("        -reset1           Reset counter 1\n");
        printf("        -debug            Activate debug mode\n");
        printf("Example : %s -p:1 -d:147 -a1:25 -a2:203\n",params[0]);
        printf("\n");
        printf("Output : (timestamp);(digital);(analog 1);(analog 2);(counter
1);(counter 2)\n");
        printf("Note : timestamp is the number of msec when data is read since
program start\n");
        printf("Example : 499;16;128;230;9;8\n");
        printf("499 : Measure done 499 msec after program start\n");

}


/*
        Read arguments, and store values
        Return true if arguments are valid
                else return false
*/
int read_param(int argc,char *params[])
```

```
{
        int erreurParam = false;
        int i;

        ipid = 0;

        for (i=1; i<argc;i++)
        {
           if ( !strncmp(params[i],"-p:",3) &&
                      !Convert_StringToInt(params[i]+3,&ipid) ) erreurParam = true;
                else
                   if ( !strncmp(params[i],"-a1:",4)    &&
                   !Convert_StringToInt(params[i]+4,&ia1) ) erreurParam = true;
                else
                   if ( !strncmp(params[i],"-a2:",4) &&
                   !Convert_StringToInt(params[i]+4,&ia2) ) erreurParam = true;
                else
                   if ( !strncmp(params[i],"-d:",3) &&
                   !Convert_StringToInt(params[i]+3,&id8) ) erreurParam = true;
                else
                   if ( !strncmp(params[i],"-num:",5) &&
                   !Convert_StringToInt(params[i]+5,&numread) ) erreurParam =
true;
                else
                   if ( !strncmp(params[i],"-delay:",7) &&
                   !Convert_StringToInt(params[i]+7,&delay) ) erreurParam = true;

                else
                   if ( !strncmp(params[i],"-dbt1:",6) &&
                   !Convert_StringToInt(params[i]+6,&dbt1) ) erreurParam = true;

                else
                   if ( !strncmp(params[i],"-dbt2:",6) &&
                   !Convert_StringToInt(params[i]+6,&dbt2) ) erreurParam = true;

                else
                   if ( !strcmp(params[i],"-debug") ){
                        debug = true;
                        DEBUG = true;
                        }
                else
                        if ( !strcmp(params[i],"-reset1") ) resetcnt1 = true;
                else
                        if ( !strcmp(params[i],"--help") ) {
                                display_help(params);
                                return false;
                        }

        }
```

```
	/*
		Send parameters to standart error
	*/
	if ( debug )
		fprintf(stderr,"Parameters : Card=%d Analog1=%d Analog2=%d
Digital=%d\n",ipid,ia1,ia2,id8);

	if (ipid<0 || ipid>3){
		printf("Invalid board address!\n");
		return -1;
	}


	if (erreurParam)
	{

		printf("Invalid or incomplete options\n");

		display_help(params);
		return false;
	}


	return true;
}

/*
	Give current timestamp in miliseconds
*/
inline unsigned long int time_msec ( void ) {
	struct timeval t; struct timezone tz;
	gettimeofday (&t,&tz);
	return (1000*t.tv_sec)+(t.tv_usec/1000);
}


int main (int argc,char *params[])
{
	int i,result[3];
	unsigned char d=0;
	long a1=0,a2=0;
	unsigned short c1=0, c2=0;
	unsigned long int start,mstart=0,lastcall=0;

	start = time_msec();

	/*
		Load parameters
		If parameters are valid continue
	*/

	if (read_param(argc,params))
	{
		/*
			Initialise USB system
			and enable debug mode

		if ( debug )
			usb_set_debug(2);
		*/
		/*
			Search the device
		*/
		if ( OpenDevice(ipid)<0 ) {
			printf("Could not open the k8055 (port:%d)\nPlease ensure that
the device is correctly connected.\n",ipid);
			return (-1);

		} else {

			if ( resetcnt1 )
				ResetCounter(1);

			if (debug && ((ia1!=-1)||(ia2!=-1)||(id8!=-1))) printf("Set ");

			mstart = time_msec(); // Measure start
			for (i=0; i<numread; i++) {

				if ( delay ) {
					// Wait until next measure
					while ( time_msec()-mstart < i*delay );
				}
/*				ReadAllAnalog(&a1,&a2); */
				a1=ReadAnalogChannel(1L);

				lastcall = time_msec();
				printf("%d, %d\n", (int)(lastcall-start), (int)a1 );
			}

			CloseDevice();
		}
	}
	return 0;
}
```

## Appendix B: Physical Sensor Code

```
/*
This code is developed for the physical sensor in the meeting detector system.

It is used to read the signal from Velleman k8055 interface board and send data to logical entity via a UDP socket. It is based on the Linux k8055 library from Sven Lindberg, the UDP client code from Professor G. Q. Maguire Jr. and physical sensor code from Daniel Hübinette.

Last updated: July 30, 2008

//Command to compile this program
gcc -o readsignal -lusb -L/usr/lib -lm -lk8055 readsignal.c
*/


/* Includes */
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

/* Function for reading from the K8055 board */
#include "k8055.h"

/* IP addresses and ports for the sensor system */
#include "address.h"

#define Maximum_Number_of_Bytes_to_read 10240

int ipid = 0;

int main(int argc, char *argv[]){

    int running = 1;

    /* Number of Bytes to read */
    int Number_of_Bytes_to_read = 16;
    /* Set how long to sleep before sampling, this is correlated to the data rate.
         Default = 3000000
    */
    long int sleepratetimeout = 3000000;

    /* Set when to sample(sleeprate = sleepratetimeout = sample straight away)
         Default = 3000000
    */
    long int sleeprate = 3000000;

    /* Client UDP port setup variables */
    int client_socket_fd;          /* Socket to client, server */
    struct sockaddr_in server_addr ;/* server's address */

    int sendto_flags = 0;

    int count;
    int i;    /* Counter for FOR loop */
    int coi; /* Counter for arg for loop */

    unsigned char bigBuffer[Maximum_Number_of_Bytes_to_read+2];
    unsigned char bigBuffer2[Maximum_Number_of_Bytes_to_read+2];
    long sample = 0;
    long sample2 = 0;

    /* Read commandline arguments to set session size
        (only between 1 and Maximum_Number_of_Bytes_to_read), else a default is set
*/
    if (argc == 2 ){
        if ((atoi(argv[1]) >= 1) && (atoi(argv[1]) <=
Maximum_Number_of_Bytes_to_read)){
            Number_of_Bytes_to_read = atoi(argv[1]);
            printf("new session size set = %d\n",atoi(argv[1]));
        }
        else{
            printf("session size = %d\n",Number_of_Bytes_to_read);
        }
    }

    if (argc == 3 ){
        if ((atoi(argv[1]) >= 1) && (atoi(argv[1]) <= 200)){
            Number_of_Bytes_to_read = atoi(argv[1]);
            printf("new session size set = %d\n",atoi(argv[1]));
            sleeprate = (long int)atoi(argv[2]);
            sleepratetimeout = sleeprate;
        }
        else{
            printf("session size = %d\n",Number_of_Bytes_to_read);
        }
    }

    /* Setup the UDP client port */
    /* Create a UDP socket */
    if ((client_socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
{
        perror("Unable to open socket");
        exit(1);
    };

    /* Initialize the server address structure */
    memset( (char*)&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family=AF_INET;
    server_addr.sin_port=htons(destination_host_port);
    if (inet_aton(destination_host,
                    (struct sockaddr*)&server_addr.sin_addr
                    ) == 0){
        fprintf(stderr, "could not get an address for: %s", destination_host);
        exit(1);
    }


    /* Initialise USB system and enable debug mode
          if ( debug )
          usb_set_debug(2);
    */

    /*
        Search the k8055 interface device
    */
    if ( OpenDevice(ipid)<0 ) {
            printf("Could not open the k8055 (port:%d)\nPlease ensure that the device is
correctly connected.\n",ipid);
            return (-1);
    }

    while (running)
        {
            sleeprate++;
            if (sleeprate >= sleepratetimeout) {

              /* Read from board */
                for (i=0; i < Number_of_Bytes_to_read; i++) {
                    sample=ReadAnalogChannel(1L);     /* sample from Channel 1 */
                    sample2=ReadAnalogChannel(2L);          /* sample from Channel 2*/
                    /*       fprintf(stdout, "read[%d]:%lu %lu\n", i,sample,sample2); */
                    bigBuffer[i]=(unsigned int)(sample & 0xff);
                    bigBuffer2[i]=(unsigned int)(sample2 & 0xff);
                }
              /* Send data to logical entity */
                /* for Channel 1 */
                bigBuffer[Number_of_Bytes_to_read]=0;
                bigBuffer[Number_of_Bytes_to_read+1]=0;
                if ((sendto(client_socket_fd,
                            bigBuffer,
                            Number_of_Bytes_to_read+2,
                            sendto_flags,
                            (struct sockaddr*)&server_addr,
                            sizeof(server_addr)
                       )
                     ) == -1) {
                        perror("Unable to send to socket");
                        close(client_socket_fd);
                        exit(1);
                }
                /* for Channel 2 */
                bigBuffer2[Number_of_Bytes_to_read]=1;
                bigBuffer2[Number_of_Bytes_to_read+1]=0;
                if ((sendto(client_socket_fd,
                            bigBuffer2,
                            Number_of_Bytes_to_read+2,
                            sendto_flags,
                            (struct sockaddr*)&server_addr,
                            sizeof(server_addr)
                       )
                     ) == -1) {
                        perror("Unable to send to socket");
                        close(client_socket_fd);
                        exit(1);
                }

             sleeprate=0;
             } /* End IF SLEEP RATE */

          } /* End WHILE */

      close(client_socket_fd);     /* close the socket */

  } /* End MAIN */
```

## Appendix C: Logical Sensor Code

```
/*
This code is developed for the logical sensor in the meeting detector context-aware
system. It is based on the server socket code from Professor G. Q. Maguire Jr. and
logical sensor code in occupancy system from Daniel Hubinette.

Last updated: October 13, 2008

//Command to compile this program
gcc -o logical -lusb -L/usr/lib -lm -lk8055 logical.c
*/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/select.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <math.h>
#include "address.h"

#define UDP_SIZE 2000

/* Definitions for the PUBLISH message */
#define Branch_label "z9hG4bK6sJ8J0y"
#define Tag_label "6sJ8"
#define Call_ID      "288"
#define Max_Forwards 10
#define Expire_time 5
#define Publish_Content_Type "application/pidf+xml"
#define PIDF_Tuple_ID "6sJ8J0"
#define Fixed_PIDF_Content2 ""
#define PIDF_event_type "occupancy"

/* Numbers of sensors used in the system */
#define Number_of_Areas 2

#define Nx 30                /* N for X */
#define Ny 30
#define Nz 30
```

```
/* Clear a buffer */
void ClearBuffer(char *msg, int size)
{ int j;
   for(j=0; j<size; j++)
      {
         msg[j]=0;
      }
}


int main(int argc, char *argv[]){


   /* INITIALIZATION VALUES */

   /* SERVER SOCKET to RECEIVE RAW DATA */
   int other_addr_len;
   /* Socket to client, server */
   int client_socket_fd;
   /* client's address */
   struct sockaddr_in client_addr;
   /* other party's address */
   struct sockaddr_in other_addr;
   /* Read buffer for received UDP packet */
   char bigBuffer[bigBufferSize];
   int sendto_flags = 0;

   /* CLIENT SOCKET to SEND PUBLISH */
   int client_socket_fd2;                          /* Socket to client, server */
   struct sockaddr_in server_addr2 ;               /* server's address */
   char bigBuffer_publish[bigBufferSize];         /* Buffer for publish*/
   int sendto_flags2 = 0;

   /*SELECT STUFF*/
   fd_set selset;                   /* Socket file descriptors we want to wake up for, using
select() */
   struct timeval timeout;    /* Timeout for select */
   int selreturn;                    /* Stores return value from Select */

   /* FILE STUFF */
   FILE *fp_file, *fp_file2;
   /* IF WE WANT TO LOG RAW DATA TO FILE SET TO 1 for YES, 0 for NO */
   int filelogging = 0;
   /* IF WE WANT TO LOG VOLTAGE DATA AND VARIANCE TO DATALOG 1 for
YES */
   int datalogging = 0;
   /* RESET EVERY 10 TIMES, USED FOR FLUSH TO FILE */
   int fcounter = 0;
   /* FIRST LINE IN LOGFILE TELLS THE NUMBER OF SAMPLES PER UDP
PACKET */
   int firsttime = 1;

   /* STATISTICS STUFF */
   int temp;                /* TEMP Variable for STATISTICS for loop */
   int prec = 2;          /* Specification of the FLOAT precision displayed in the text */
   int prec2 = 6;
   int intval[Number_of_Areas];      /* Sample Value between 0-255 */
   int  n,  nn1  = 0;                    /* Number of samples*/
   int i, j;                        /* Index for iteration */
   int bytes_read;                 /* Number of bytes read from the socket */

   int corrdatacount[Number_of_Areas];
   float voltage[Number_of_Areas]; /* VOLTAGE of one sample */
   float sumvol[Number_of_Areas];    /* The sum of the voltage samples     =
(i=1:n)Sum(Xi) */
   float sumvol2[Number_of_Areas]; /* The sum^2 of the voltage samples =
(i=1:n)Sum(Xi^2) */
   /* The average voltage of the RAW DATA SAMPLE = 1/n*(i=1:n)Sum(Xi) */
   float sumave[Number_of_Areas];
   /* The average voltage of the RAW DATA SAMPLE = 1/n*(((i=1:n)Sum(Xi))^2) */
   float sum2ave[Number_of_Areas];
   float variance[Number_of_Areas]; /* The variance */
   float deviance[Number_of_Areas]; /* The deviance */
   float dv[Number_of_Areas],v1[Number_of_Areas],v2[Number_of_Areas];
   float maxval[Number_of_Areas];      /* Maximum RAW DATA Value Seen */
   float minval[Number_of_Areas];      /* Minimum RAW DATA Value Seen */
   float  sumy[Number_of_Areas];          /*  SUMY */
   float  sumy2[Number_of_Areas];        /* SUMYCUBE */
   float cubesumy[Number_of_Areas]; /* CUBED SUM */
   float corrdata[Number_of_Areas][Nx] = {0}; /* Initialization of corrdata */
   float corr2data[Number_of_Areas][Nx] = {0};
   float sensordata[Number_of_Areas][Nx];
   float sumx[Number_of_Areas];
   float sumx2[Number_of_Areas];
   float cubesumx[Number_of_Areas];
   float  threshold  = 0.8;                /*  Threshold  for  correlation  detection  */
   float rho[Number_of_Areas];
   float rho2[Number_of_Areas];
   float sumxiyi[Number_of_Areas];
   float sumxizi[Number_of_Areas];
   float sumxsumydnx[Number_of_Areas];
   float sumxsumzdnx[Number_of_Areas];
   float deviancey[Number_of_Areas];
   float deviancex[Number_of_Areas];
   float deviancez[Number_of_Areas];
   float sumz[Number_of_Areas];
   float sumz2[Number_of_Areas];
   float cubesumz[Number_of_Areas];
```

```
/* DETECTION STUFF */

/* Definitions of boundaries for state based detection */
float topzone = 28;
float midtop    = 23;
float midbot    = 16;
float botzone = 10;

/* Count for state based detection */
int comefromright[Number_of_Areas];
int comefromleft[Number_of_Areas];

/* Count for correlation based detection */
int personsinarea[Number_of_Areas];
int corrpersons[Number_of_Areas];
int prevcorr[Number_of_Areas];

int current_area;    /* Change for area */

/* Reset if exceeds */
int rightresetcount[Number_of_Areas];
int leftresetcount[Number_of_Areas];
int detectreset= 15;

double convertvar = 0;

/* NOTIFY CONTEXT SERVER */
int notify = 0;
int corrnote = 0;

int coi;
int foi;

char pidf_core[UDP_SIZE*sizeof(char)];
char * PIDF_Note;
char * sensor_id;
char * Destination_Machine_Address = destination_host;
int SIP_Default_Port_Number =    presence_server_host_port;

/* Initialization of values */
for (i=0; i < Number_of_Areas; i++) {
        personsinarea[Number_of_Areas]=0;
        corrpersons[Number_of_Areas]=0;
        prevcorr[Number_of_Areas]=0;
        corrdatacount[Number_of_Areas] = 0;
        voltage[Number_of_Areas] = 0;
        sumvol[Number_of_Areas] = 0;
        sumvol2[Number_of_Areas] = 0;
        sumave[Number_of_Areas] = 0;
        sum2ave[Number_of_Areas] = 0;
        deviance[Number_of_Areas] = 0;
        deviance[Number_of_Areas] = 0;
        dv[Number_of_Areas],v1[Number_of_Areas],v2[Number_of_Areas] = 0;
        maxval[Number_of_Areas] = 0;
        minval[Number_of_Areas] = 2;
        comefromright[Number_of_Areas] = 0;
        comefromleft[Number_of_Areas] = 0;
        rightresetcount[Number_of_Areas] = 0;
        leftresetcount[Number_of_Areas] = 0;

  }

  /* START OF LOGIC CODE */
  sendto_flags=0;

  for(coi = 0; coi < argc; coi++){
    printf("arg %d: %s\n", coi, argv[coi]);
  }

  if (argc == 2 ){
    if ((strtod(argv[1],NULL) >= -1) && (strtod(argv[1],NULL) <= 1)){
        convertvar = strtod(argv[1],NULL);
        threshold = convertvar;
        printf("new threshold set = %f\n",threshold);

    }
    else{
        printf("threshold size = %f\n",threshold);
    }
  }


/* Correlation based detection for sensor 0 */

/* Correlation Data 1 for exit */
  corrdata[0][0] = 16.56;
  corrdata[0][1] = 18.56;
  corrdata[0][2] = 26.94;
  corrdata[0][3] = 33.88;
  corrdata[0][4] = 33.31;
  corrdata[0][5] = 7.38;
  corrdata[0][6] = 0.00;
  corrdata[0][7] = 0.00;
  corrdata[0][8] = 0.00;
  corrdata[0][9] = 11.00;
  corrdata[0][10] = 25.62;
  corrdata[0][11] = 31.06;
  corrdata[0][12] = 32.12;
  corrdata[0][13] = 28.88;
  corrdata[0][14] = 26.88;
  corrdata[0][15] = 24.50;
  corrdata[0][16] = 22.81;
  corrdata[0][17] = 20.38;
  corrdata[0][18] = 18.69;
  corrdata[0][19] = 19.06;
  corrdata[0][20] = 18.38;
  corrdata[0][21] = 17.44;
  corrdata[0][22] = 16.75;
  corrdata[0][23] = 16.81;
  corrdata[0][24] = 16.31;
  corrdata[0][25] = 16.38;
  corrdata[0][26] = 16.62;
  corrdata[0][27] = 16.25;
  corrdata[0][28] = 16.56;
  corrdata[0][29] = 16.12;

  /* Correlation Data 2 for entry */
  corr2data[0][0] = 18.69;
  corr2data[0][1] = 18.19;
  corr2data[0][2] = 18.50;
  corr2data[0][3] = 17.12;
  corr2data[0][4] = 16.19;
  corr2data[0][5] = 14.75;
  corr2data[0][6] = 14.06;
  corr2data[0][7] = 0.69;
  corr2data[0][8] = 0.00;
  corr2data[0][9] = 0.00;
  corr2data[0][10] = 15.31;
  corr2data[0][11] = 34.25;
  corr2data[0][12] = 33.31;
  corr2data[0][13] = 34.19;
  corr2data[0][14] = 33.19;
  corr2data[0][15] = 33.31;
  corr2data[0][16] = 22.25;
  corr2data[0][17] = 15.12;
  corr2data[0][18] = 7.25;
  corr2data[0][19] = 3.00;
  corr2data[0][20] = 6.12;
  corr2data[0][21] = 9.12;
  corr2data[0][22] = 13.50;
  corr2data[0][23] = 16.00;
  corr2data[0][24] = 18.00;
  corr2data[0][25] = 19.25;
  corr2data[0][26] = 20.06;
  corr2data[0][27] = 20.44;
  corr2data[0][28] = 20.25;
  corr2data[0][29] = 20.62;

/* Correlation based detection for sensor 1 */
/* Correlation Data 1 for exit */
  corrdata[1][0] = 18.900000;
  corrdata[1][1] = 19.366667;
  corrdata[1][2] = 20.533333;
  corrdata[1][3] = 22.333334;
  corrdata[1][4] = 28.833334;
  corrdata[1][5] = 35.466667;
  corrdata[1][6] = 35.366665;
  corrdata[1][7] = 35.166668;
  corrdata[1][8] = 27.333334;
  corrdata[1][9] = 0.000000;
  corrdata[1][10] = 1.000000;
  corrdata[1][11] = 0.000000;
  corrdata[1][12] = 1.000000;
  corrdata[1][13] = 18.333333;
  corrdata[1][14] = 25.866667;
  corrdata[1][15] = 23.433334;
  corrdata[1][16] = 21.566668;
  corrdata[1][17] = 18.700001;
  corrdata[1][18] = 15.833334;
  corrdata[1][19] = 13.966666;
  corrdata[1][20] = 14.566667;
  corrdata[1][21] = 14.400000;
  corrdata[1][22] = 15.066667;
  corrdata[1][23] = 15.233334;
  corrdata[1][24] = 16.366667;
  corrdata[1][25] = 16.433332;
  corrdata[1][26] = 17.166666;
  corrdata[1][27] = 18.000000;
  corrdata[1][28] = 19.733334;
  corrdata[1][29] = 20.033333;


  /* Correlation Data 2 for entry */
  corr2data[1][0] = 18.799999;
  corr2data[1][1] = 19.200001;
  corr2data[1][2] = 18.833334;
  corr2data[1][3] = 13.833333;
  corr2data[1][4] = 0.000000;
  corr2data[1][5] = 0.000000;
  corr2data[1][6] = 0.000000;
  corr2data[1][7] = 0.000000;
  corr2data[1][8] = 35.466667;
  corr2data[1][9] = 35.599998;
  corr2data[1][10] = 35.500000;
  corr2data[1][11] = 35.090000;
  corr2data[1][12] = 35.060000;
  corr2data[1][13] = 22.866668;
  corr2data[1][14] = 7.366666;
```

```
   corr2data[1][15] = 2.000000;
   corr2data[1][16] = 0.000000;
   corr2data[1][17] = 2.533333;
   corr2data[1][18] = 5.066667;
   corr2data[1][19] = 8.233334;
   corr2data[1][20] = 12.166666;
   corr2data[1][21] = 15.300000;
   corr2data[1][22] = 18.433333;
   corr2data[1][23] = 20.066666;
   corr2data[1][24] = 21.366667;
   corr2data[1][25] = 22.333333;
   corr2data[1][26] = 22.800000;
   corr2data[1][27] = 23.366667;
   corr2data[1][28] = 23.600000;
   corr2data[1][29] = 23.533333;


        printf("Setting up the Correlation 1 Dataset\n");

        for (current_area=0; current_area < Number_of_Areas; current_area++){

            for(foi = 0; foi < Ny; foi++){
                    printf("%f\n",corrdata[current_area][foi]);
                    sumy[current_area] = sumy[current_area] +
corrdata[current_area][foi];
                    sumy2[current_area] = sumy2[current_area] +
(corrdata[current_area][foi] * corrdata[current_area][foi]);
                    cubesumy[current_area] = (sumy2[current_area] *
sumy2[current_area]);
            }

/*          printf("sumy[%d]=%f\n", current_area, sumy[current_area]);
            printf("sumy2[%d]=%f\n", current_area, sumy2[current_area]);
*/
            printf("Setting up the Correlation 2 Dataset\n");
            for(foi = 0; foi < Ny; foi++){
                    printf("%f\n",corr2data[current_area][foi]);
                    sumz[current_area] = sumz[current_area] +
corr2data[current_area][foi];
                    sumz2[current_area] = sumz2[current_area] +
(corr2data[current_area][foi] * corr2data[current_area][foi]);
                    cubesumz[current_area] = (sumz2[current_area] *
sumz2[current_area]);
            }

/*          printf("sumz[%d]=%f\n", current_area, sumz[current_area]);
            printf("sumz2[%d]=%f\n", current_area, sumz2[current_area]);
*/
            for (j=0; j < Nx; j++) {
                    sensordata[current_area][j] = 0.0;
        }

    }


   /* Create a UDP server socket */

   client_socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
   /* printf("%d : FD SOCKET",client_socket_fd); */
   if (client_socket_fd == -1) {
      perror("Unable to open socket");
      exit(1);
   };


   /* Initialize the server address structure */
   memset( (char*)& client_addr, 0, sizeof(client_addr));
   client_addr.sin_family=AF_INET;
   client_addr.sin_port=htons(destination_host_port);
   client_addr.sin_addr.s_addr = htonl(INADDR_ANY);

   if (bind(client_socket_fd,
              (struct sockaddr*)&client_addr,
              sizeof(client_addr))==-1) {
      close(client_socket_fd);
      exit(1);
   }


   /* CREATE A CLIENT UDP SOCKET FOR PUBLISH */

   /* SETUP THE UDP CLIENT PUBLISH PORT */

   /* Create a UDP socket */
   if ((client_socket_fd2 = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -
1) {
      perror("Unable to open socket");
      exit(1);

   };

   /* Initialize the server address structure */
   memset( (char*)&server_addr2, 0, sizeof(server_addr2));
   server_addr2.sin_family=AF_INET;
   server_addr2.sin_port=htons(presence_server_host_port);

   if (inet_aton(presence_server_host,
                 (struct in_addr*)&server_addr2.sin_addr) == 0) {
      fprintf(stderr, "could not get an address for: %s", presence_server_host);
      exit(1);
```

```
}

/* CHECK IF WE WANT TO LOG */
if (filelogging == 1) {
   /* OPEN LOG FILE FOR RAW DATA */
   if((fp_file=fopen("rawdatastream", "a"))==NULL) {
      printf("Cannot open file. Will not log data\n");
      filelogging=0;
   }
}

if (datalogging == 1) {
   /* OPEN LOG FILE FOR RAW DATA */
   if((fp_file2=fopen("datalog", "a"))==NULL) {
      printf("Cannot open file. Will not log data\n");
      datalogging=0;
   }
}


/* READ DATA INDEFINATELY */
while(1)
   {

      /* SELECT STUFF*/
      /* FD_ZERO() clears out the fd_set called selset, so that
        it doesn't contain any file descriptors. */
      FD_ZERO(&selset);

      /* FD_SET() adds the file descriptor "client_socket_fd" to the fd_set,
        so that select() will return if a connection comes in
        on that socket (which means you have to do accept(), etc. */
      FD_SET(client_socket_fd, &selset);

      /* Timeout specification */
      /* This must be reset every time select() is called */
      timeout.tv_sec  =1;           /*  timeout  (secs.)  */
      timeout.tv_usec  = 0;         /*  0  microseconds  */


      /* CHECK FDs ATT REGULAR INTERVALS */

      selreturn = select(client_socket_fd+1,
                        &selset,NULL,NULL, &timeout);

      /* IF NO FDs DATA IS RECEIVED */
      if (selreturn == 0) {
      printf("No data being received from sensor!\n");
      continue;            /* Go around the outer while loop once again */
      }
```

```
        /* IF DATA IS RECEIVED */
        if (selreturn>0){

          bytes_read = recvfrom(client_socket_fd,
                          bigBuffer,
                          bigBufferSize,
                          sendto_flags,
                          (struct sockaddr*)&other_addr,
                                  &other_addr_len);
        if (bytes_read == -1) {
           perror("Unable to receive from socket");
           close(client_socket_fd);
           exit(1);
        }

           /* Get the area id from the buffer data */
        current_area = bigBuffer[bytes_read-2];

        fprintf(stderr, "The current area is [%d]----------------------\n", current_area);


        /* WRITE OUT RAW DATA WITH INTERESTING FACTS ABOUT DATA
SOURCE */

        /*printf("Received packet from %s:%d\nData: %s\nString length=%d\n",
           inet_ntoa(other_addr.sin_addr),
           ntohs(other_addr.sin_port),
           bigBuffer,
           strlen(bigBuffer));
        */

        /* PRINT OUT INTERESTING FACTS ABOUT DATA SOURCE */

        /*printf("Received packet from %s:%d\nlength=%d\n",
           inet_ntoa(other_addr.sin_addr),
           ntohs(other_addr.sin_port),
           strlen(bigBuffer));
        */

        /* GET NUMBER OF BYTES FOR FILE LOG */
        /*if ((filelogging == 1) &&(firsttime == 1)){

        fprintf(fp_file,"SAMPLES %d \n", strlen(bigBuffer));
        firsttime = 0;
        }*/

        /* STATISTICS: DO STUFF WITH RETRIEVED DATA */

        /* RESET TEMPORARY STATISTICAL DATA */
        sumave[current_area] = 0;
        sumvol[current_area]  =0;
        sumvol2[current_area] = 0;
        sum2ave[current_area] = 0;
        variance[current_area] = 0;
        deviance[current_area] = 0;
        sumx[current_area] = 0;
        sumxiyi[current_area] = 0;
        sumxizi[current_area] = 0;
        sumx2[current_area] = 0;
        cubesumx[current_area] = 0;

        /* Calculate the average voltage & voltage^2 from group of samples */
        for(temp = 0; temp < (bytes_read-2); temp++){
          intval[current_area] = (int)(unsigned char) bigBuffer[temp];

          if (filelogging == 1){

            fprintf(fp_file,"%d ", intval[current_area]);

            /* FLUSH EVERY 8 WRITES TO MAKE SURE DATA IS WRITTEN TO
FILE */
            fcounter++;
            if (fcounter > 8) {
              fflush(fp_file);
              fcounter = 0;
            }
          }

          voltage[current_area] = intval[current_area];

            /* GET THE SUM OF VOLTAGES */
          sumvol[current_area] = sumvol[current_area] + voltage[current_area];

          /* GET THE SUM VOLTAGES^2 */
          sumvol2[current_area] = sumvol2[current_area] + (voltage[current_area]
* voltage[current_area]);

        } /* END FOR LOOP */

        /* Write NEWLINE TO LOG FILE AFTER EVERY N Number of SAMPLES */
        if (filelogging == 1)
          {
            fprintf(fp_file,"\n");

                }
          n = bytes_read-2;
        nm1 = n-1;
        /* MAKE SURE WE DON'T DIVIDE BY ZERO WHEN GETTING THE SUM
AVERAGE */
        if ((n != 0) && (nm1 != 0)){
           sumave[current_area] = sumvol[current_area]/n;
              sum2ave[current_area] = (sumvol[current_area] *
sumvol[current_area])/n;
           variance[current_area] = (sumvol2[current_area] -
sum2ave[current_area])/nm1;
              deviance[current_area]  = sqrtf(variance[current_area]);
        }

           v2[current_area] = sumave[current_area];
        dv[current_area] = v2[current_area] - v1[current_area];
           v1[current_area] = v2[current_area];

        /* SAVE THE MAX VALUE SEEN */
        if (sumave[current_area] > maxval[current_area])
              maxval[current_area] = sumave[current_area];

        /* SAVE THE MIN VALUE SEEN */
        if (sumave[current_area] < minval[current_area]) {
           minval[current_area] = sumave[current_area];
           if (minval[current_area] < 0)
                 minval[current_area]=0;
        }

        /* PRINT OUT FACTS TO SCREEN */
           /* printf("Bytes per Packet = %d\n",n);
            printf("Mean          value    =  %*f \n",prec,sumave[current_area]);
            printf("Variance    value    =  %*f \n",prec2,variance[current_area]);
            printf("Deviance    value    =  %*f \n",prec,deviance[current_area]);
            printf("dvdt                        =  %*f \n",prec,dv[current_area]);
            printf("Minimum      value    =  %*f \n",prec,minval[current_area]);
            printf("Maximum      value    =  %*f \n",prec,maxval[current_area]);
           */

        /* PRINT OUT FACTS TO DATALOG */

        if (datalogging == 1)
           {
              fprintf(fp_file2,"%f,%f \n",sumave[current_area],
variance[current_area]);
                 fflush(fp_file2);
           }


        /* DETECTION START */

        /* CORRELATION DETECTION START */
        /* Shift Data and store latest value */
        sensordata[current_area][0] = sensordata[current_area][1];
        sensordata[current_area][1] = sensordata[current_area][2];
        sensordata[current_area][2] = sensordata[current_area][3];
        sensordata[current_area][3] = sensordata[current_area][4];
        sensordata[current_area][4] = sensordata[current_area][5];
        sensordata[current_area][5] = sensordata[current_area][6];
        sensordata[current_area][6] = sensordata[current_area][7];
        sensordata[current_area][7] = sensordata[current_area][8];
        sensordata[current_area][8] = sensordata[current_area][9];
        sensordata[current_area][9] = sensordata[current_area][10];
        sensordata[current_area][10] = sensordata[current_area][11];
        sensordata[current_area][11] = sensordata[current_area][12];
        sensordata[current_area][12] = sensordata[current_area][13];
        sensordata[current_area][13] = sensordata[current_area][14];
        sensordata[current_area][14] = sensordata[current_area][15];
        sensordata[current_area][15] = sensordata[current_area][16];
        sensordata[current_area][16] = sensordata[current_area][17];
        sensordata[current_area][17] = sensordata[current_area][18];
        sensordata[current_area][18] = sensordata[current_area][19];
        sensordata[current_area][19] = sensordata[current_area][20];
        sensordata[current_area][20] = sensordata[current_area][21];
        sensordata[current_area][21] = sensordata[current_area][22];
        sensordata[current_area][22] = sensordata[current_area][23];
        sensordata[current_area][23] = sensordata[current_area][24];
        sensordata[current_area][24] = sensordata[current_area][25];
        sensordata[current_area][25] = sensordata[current_area][26];
        sensordata[current_area][26] = sensordata[current_area][27];
        sensordata[current_area][27] = sensordata[current_area][28];
        sensordata[current_area][28] = sensordata[current_area][29];
        sensordata[current_area][29] = sumave[current_area];

        if (corrdatacount[current_area] == 30) {printf("Sensor Data Populated\n");
        }
        if (corrdatacount[current_area] < 30) {
            corrdatacount[current_area] = corrdatacount[current_area] +1;
            printf("Populating Sensor Data\n");
        }

        /* SUMXiYi */
        for (coi = 0; coi < Nx; coi++){
                printf("Sensor[%d] Data = %.*f : Correlation Data = %.*f : Correlation
Data2 = %.*f\n",
                        current_area,
                        prec,sensordata[current_area][coi],
                        prec,corrdata[current_area][coi],
                        prec,corr2data[current_area][coi]);
            sumxiyi[current_area] = sumxiyi[current_area] +
(sensordata[current_area][coi] * corrdata[current_area][coi]);
```

```
            sumxizi[current_area] = sumxizi[current_area] +
(sensordata[current_area][coi] * corr2data[current_area][coi]);
            sumx[current_area] = sumx[current_area] +
sensordata[current_area][coi];
            sumx2[current_area] = sumx2[current_area] +
(sensordata[current_area][coi] * sensordata[current_area][coi]);
            cubesumx[current_area] = (sumx2[current_area] * sumx2[current_area]);

         }

         sumxsumydnx[current_area] = ((sumx[current_area]
*sumy[current_area])/Nx);
         sumxsumzdnx[current_area] = ((sumx[current_area]
*sumz[current_area])/Nx);

         deviancex[current_area] = sqrt(sumx2[current_area] -
((sumx[current_area]*sumx[current_area])/Nx));
         deviancey[current_area] = sqrt(sumy2[current_area] -
((sumy[current_area]*sumy[current_area])/Ny));
         deviancez[current_area] = sqrt(sumz2[current_area] -
((sumz[current_area]*sumz[current_area])/Nz));

         rho[current_area] = (sumxiyi[current_area] -
sumxsumydnx[current_area])/(deviancex[current_area]*deviancey[current_area]);
            rho2[current_area] = (sumxizi[current_area] -
sumxsumzdnx[current_area])/(deviancex[current_area]*deviancez[current_area]);

            /* printf("Sumxiyi    = %f\n",sumxiyi[current_area]);
             printf("Sumxizi    = %f\n",sumxizi[current_area]);
             printf("Sumx       = %f\n",sumx[current_area]);
             printf("Sumx2      = %f\n",sumx2[current_area]);
             printf("Cubesumx = %f\n",cubesumx[current_area]);
             printf("Sumy       = %f\n",sumy[current_area]);
             printf("Sumy2      = %f\n",sumy2[current_area]);
             printf("Cubesumy = %f\n",cubesumy[current_area]);
             printf("Sumz       = %f\n",sumz[current_area]);
             printf("Sumz2      = %f\n",sumz2[current_area]);
             printf("Cubesumz = %f\n",cubesumz[current_area]);
             printf("Nx         = %d\n",Nx);
             printf("Ny         = %d\n",Ny);
             printf("Nz         = %d\n",Nz);
             printf("DevianceX= %f\n",deviancex[current_area]);
             printf("DevianceY= %f\n",deviancey[current_area]);
             printf("DevianceZ= %f\n",deviancez[current_area]);
             printf("Rho        = %f\n",rho[current_area]);
             printf("Rho2       = %f\n",rho2[current_area]);
         */

         /* Correlation coefficient comparison for exit */
         if (rho[current_area] >= threshold) {
                 printf("RIGHT TO LEFT DETECTED\n");
                 if (corrpersons[current_area] >= 1) {
                     corrpersons[current_area] = corrpersons[current_area] - 1;
                     cornote = 1;
                 }
         }

         /* Correlation coefficient comparison for entry */
         if (rho2[current_area] >= threshold){
                 printf("LEFT TO RIGHT DETECTED 2\n");
                 corrpersons[current_area] = corrpersons[current_area] + 1;
                 corrnote = 1;

         }

         /* STATE BASED DETECTION */
         if ((comefromleft[current_area]==0) && (comefromright[current_area] ==
0)) {
            printf("STATE 0 \n");
            if ((sumave[current_area] >=    midtop ) &&
(personsinarea[current_area] > 0)) {
                 comefromright[current_area]=1; /* SET STATE 1 - PASS TOPZONE */
            }
            if (sumave[current_area] <= midbot) {
                 comefromleft[current_area]=1;    /* SET STATE 1 - PASS BOTZONE */
            }

         }

         if (comefromright[current_area] == 1) {
            printf("STATE 1 \n");
            if (sumave[current_area] >= topzone) {
                 comefromright[current_area] = 2; /* SET STATE 2 - PASS MIDTOP */
                 printf("PERSON COMING FROM THE RIGHT SIDE <-----------\n");
                 rightresetcount[current_area] =0;
            }
            rightresetcount[current_area]++;
         }

         else if (comefromleft[current_area] == 1) {
            printf("STATE 1 \n");
            if (sumave[current_area] <= botzone) {
                 comefromleft[current_area] = 2;    /* SET STATE 2 - PASS MIDBOT */
                 printf("PERSON COMING FROM THE LEFT SIDE ------------>\n");
                 leftresetcount[current_area] = 0;
            }
            leftresetcount[current_area]++;
         }

         if (comefromright[current_area] == 2) {
```

```
   printf("STATE 2 \n");
   if (sumave[current_area] <= midtop) {
      comefromright[current_area] = 3; /* SET STATE 3 - PASS MIDBOT */
      rightresetcount[current_area] =0;
   }
   rightresetcount[current_area]++;
}

else if (comefromleft[current_area] == 2) {
   printf("STATE 2 \n");

   if (sumave[current_area] >= midbot) {
      comefromleft[current_area] = 3;    /* SET STATE 3 - PASS MIDTOP */
      leftresetcount[current_area] = 0;
   }
   leftresetcount[current_area]++;
}

if (comefromright[current_area] == 3) {
   printf("STATE 3 \n");

   if (sumave[current_area] <= midbot) {
      comefromright[current_area] = 4; /* SET STATE 4 - PASS BOTZONE */
      rightresetcount[current_area] =0;
   }
   rightresetcount[current_area]++;
}

else if (comefromleft[current_area] == 3) {
   printf("STATE 3 \n");

   if (sumave[current_area] >= midtop) {
      comefromleft[current_area] = 4;    /* SET STATE 4 - PASS TOPZONE */
      leftresetcount[current_area] = 0;
   }
   leftresetcount[current_area]++;
}

if (comefromright[current_area] == 4) {
   printf("STATE 4 \n");

   if (sumave[current_area] <= botzone) {
      comefromright[current_area] = 5; /* SET STATE 5 - PASS MIDBOT */
      rightresetcount[current_area] =0;
   }
   rightresetcount[current_area]++;
}

else if (comefromleft[current_area] == 4) {
   printf("STATE 4 \n");
```

```
            if (sumave[current_area] >= topzone) {
                comefromleft[current_area] = 5; /* SET STATE 5    - PASS MIDTOP */
                leftresetcount[current_area] = 0;
            }
            leftresetcount[current_area]++;
        }

        if (comefromright[current_area] == 5) {
            printf("STATE 5 \n");

            if (sumave[current_area] >= midbot) {
                printf("EXIT DETECTED\n");
                personsinarea[current_area]--;
                notify = 1;
                comefromright[current_area] = 0; /* SET STATE 0 */
            }
            rightresetcount[current_area]=0;
        }

        else if (comefromleft[current_area] == 5) {
            printf("STATE 5 \n");

            if (sumave[current_area] <= midtop) {
                printf("ENTRY DETECTED\n");
                personsinarea[current_area]++;
                comefromleft[current_area] = 0; /* SET STATE 0 */
                notify = 1;
            }
            leftresetcount[current_area]=0;
        }

        /* TIMEOUTS */
        if (rightresetcount[current_area] > detectreset)
comefromright[current_area]=0;
        if (leftresetcount[current_area] > detectreset) comefromleft[current_area] =
0;

        /* NOTIFY CONTEXT SERVER OF UPDATE using correlation based
detection */
        if (corrpersons[current_area] == 0) printf("Correlation : Empty in area\n");
        if (corrpersons[current_area] == 1) printf("Correlation : Individual in
area\n");
        if (corrpersons[current_area] > 1)    {
            printf("Correlation : Meeting (%d) in area\n", corrpersons[current_area]);
        }

        /* NOTIFY CONTEXT SERVER OF UPDATE using state based detection */
        /* if (personsinarea[current_area] == 0) printf("State Based : Area is
EMPTY\n");
```

```
                if (personsinarea[current_area] == 1) printf("State Based : ONE person
in area\n");
                if (personsinarea[current_area] > 1){
                      printf("State Based : MANY (%d) people in area\n",
                      personsinarea[current_area]);
                }
          */
          if ((corrnote == 1) && (prevcorr[current_area] !=
corrpersons[current_area])){
               notify = 0;
               corrnote = 0;
               prevcorr[current_area] = corrpersons[current_area];
               if (corrpersons[current_area] == 0) PIDF_Note = "Empty";
               if (corrpersons[current_area] == 1) PIDF_Note = "Individual";
               if (corrpersons[current_area] > 1) PIDF_Note = "Meeting";

               printf("Now there are %d persons in this area!\n",
personsinarea[current_area]);

               switch (current_area)
               {
                     case 0: sensor_id = "OpenArea";
                        break;
                     case 1: sensor_id = "MINT";
                        break;
               }

               sprintf(pidf_core,
               "<?xml version=\"1.0\" encoding=\"UTF-8\"?><presence
xmlns=\"urn:ietf:params:xml:ns:pidf\"entity=\"sip:%s@%s\"><tuple
id=\"%s\"><status><basic>open</basic><area>%s</area><occupancy>%s</oc
cupancy></status><note>%d</note><contact
priority=\"0.8\">ccsmoto</contact></tuple></presence>\r\n",
                        Source_Machine_Name,
                        presence_server_host,
                        PIDF_Tuple_ID,
                        sensor_id,
                        PIDF_Note,
                        corrpersons[current_area]);

          sprintf(bigBuffer_publish,
          "PUBLISH sip:%s@%s SIP/2.0\r\nVia:
SIP/2.0/UDP %s:%d;branch=%s\r\nTo: <sip:%s@%s>\r\nFrom:
<sip:%s@%s>;tag=%s\r\nCall-ID: %s@%s\r\nCSeq: 1 PUBLISH\r\nMax-
Forwards: %d\r\nExpires: %d\r\nEvent: %s\r\nContent-Type: %s\r\nContent-
Length: %d\r\n\r\n%s",
                     User_name,
                     presence_server_host,
                     Destination_Machine_Address,
                     SIP_Default_Port_Number,
                     Branch_label,
                     Source_Machine_Name,
                     presence_server_host,
                     Publish_Machine_Name,
                     Destination_Machine_Address,
                     Tag_label,
                     Call_ID,
                     Destination_Machine_Address,
                     Max_Forwards,
                     Expire_time,
                     PIDF_event_type,
                     Publish_Content_Type,
                     (int)strlen(pidf_core), pidf_core);

          /* Make a PUBLISH if occupancy information changes */

          if ((sendto(client_socket_fd2,
                      bigBuffer_publish,
                      strlen(bigBuffer_publish),
                      sendto_flags2,
                      (struct sockaddr*)&server_addr2,
                      sizeof(server_addr2))) == -1) {

               perror("Error Sending PUBLISH message. \n");
               close(client_socket_fd2);
               exit(1);
          }

          ClearBuffer(bigBuffer_publish,strlen(bigBuffer_publish));

          }

          } /* END IF SELECT */

        }     /* END WHILE  */

       close(client_socket_fd);   /* Close the socket */
     close(client_socket_fd2);
     /* IF FILE HAS BEEN OPENED */
     if (filelogging == 1) {
        fclose(fp_file); /* Close the logfile */
     }
     if (datalogging == 1) {
        fclose(fp_file2); /* Close the logfile */
     }

     return 0;
        exit(0);
}
```

## Appendix D: Header File for Main Program

This is the header file for physical and logical sensor code.

```
#define bigBufferSize 8192

#define destination_host "130.237.15.252"

#define destination_host_port 49152

#define Source_Machine_Name "ccsleft"

#define Publish_Machine_Name "ccsmoto"

#define User_name "xueliang"

#define presence_server_host "130.237.15.238"

#define presence_server_host_port 5060
```

## Appendix E: SER Configuration File

# This is the ser.cfg file for SER configuration with presence module.

```
debug=9 # debug level (cmd line: -dddddddddd)
check_via=no # (cmd. line: -v)
dns=no                  # (cmd. line: -r)
rev_dns=no          # (cmd. line: -R)
port=5060
children=2
#alias="wireless.kth.se"
mhomed=yes    # usefull for multihomed hosts, small performance penalty
#tcp_accept_aliases=yes # accepts the tcp alias via option (see NEWS)
#tcp_poll_method="sigio_rt"

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

loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/avp.so"
loadmodule "/usr/local/lib/ser/modules/avpops.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/textops.so"
loadmodule "/usr/local/lib/ser/modules/mysql.so"
loadmodule "/usr/local/lib/ser/modules/dialog.so"
loadmodule "/usr/local/lib/ser/modules/rls.so"
loadmodule "/usr/local/lib/ser/modules/pa.so"
loadmodule "/usr/local/lib/ser/modules/presence_b2b.so"
loadmodule "/usr/local/lib/ser/modules/uri.so"
loadmodule "/usr/local/lib/ser/modules/uri_db.so"
loadmodule "/usr/local/lib/ser/modules/domain.so"
loadmodule "/usr/local/lib/ser/modules/fifo.so"
loadmodule "/usr/local/lib/ser/modules/xmlrpc.so"
loadmodule "/usr/local/lib/ser/modules/xlog.so"
loadmodule "/usr/local/lib/ser/modules/msilo.so"
loadmodule "/usr/local/lib/ser/modules/xcap.so"
#loadmodule "/usr/local/lib/ser/modules/cpl-c.so"
#loadmodule "/usr/lib/ser/modules/unixsock.so"

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

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

# modparam("msilo","registrar","sip:registrar@test-domain.com")
modparam("msilo","use_contact",0)
modparam("msilo","expire_time",7200)

# -- usrloc params --

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

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

modparam("rls", "min_expiration", 200)
modparam("rls", "max_expiration", 300)
modparam("rls", "default_expiration", 300)
modparam("rls", "auth", "none")
#modparam("rls", "xcap_root", "http://localhost/xcap")
modparam("rls", "reduce_xcap_needs", 1)
modparam("rls", "db_mode", 0)
modparam("rls", "db_url", "mysql://root:helso@localhost:3306/ser")

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

```
modparam("pa", "max_subscription_expiration", 600)
# maximum value of Expires for publications
modparam("pa", "max_publish_expiration", 120)
# how often test if something changes and send NOTIFY
modparam("pa", "timer_interval", 10)

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

modparam("usrloc", "db_mode", 0)
modparam("domain", "db_mode", 1)
modparam("domain|uri_db|acc|auth_db|usrloc|msilo", "db_url",
"mysql://root:helso@localhost:3306/ser")

modparam("fifo", "fifo_file", "/tmp/ser_fifo")
modparam("xcap", "xcap_root", "http://localhost/xcap")


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

# main routing logic

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

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

```
                sl_send_reply("513", "Message too big");
                break;
        };

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

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

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


        if (uri=~"130.237.15.238") {

                #if (!lookup_domain("To")) {
                        if (lookup_domain("$fd","@from.uri.host")) {
                        xlog("L_ERR", "Unknown domain to: %tu from: %fu\n");
                        route(1);
                        break;
                }

                if (method=="REGISTER") {

                        # Uncomment this if you want to use digest authentication
                        #if (!www_authorize("130.237.15.238", "subscriber")) {
                        #       www_challenge("130.237.15.238", "0");
                        #break;
                        #};

                        save("location");

                        # dump stored messages - route it through myself (otherwise
routed via DNS!)
                        if (m_dump("sip:127.0.0.1")) {
                            xlog("L_ERR", "MSILO: offline messages for %fu
dumped\n");
                        }
                        break;
                };


                if (method=="SUBSCRIBE") {
                        if (!t_newtran()) {
                                sl_reply_error();
                                break;
                        };

                        if (@to.tag=="") {
                                # only for new subscriptions (with empty to tag)

                                if (lookup_user("To")) {
                                        # existing user -> it is subscription to PA
                                        if (handle_subscription("registrar")) {
                                                if ((@msg.event=~"presence\.winfo")) {
                                                        # new watcher info subscription
                                                        # sends one watcher info NOTIFY
message with all saved authorization requests
                                                        xlog("L_ERR", "dumping stored
winfo to %fu\n");
                                                        dump_stored_winfo("registrar",
"presence");
                                                }
                                                else {
                                                        # new presence subscription
                                                        if ((@msg.event=~"presence")
&& (%subscription_status=="pending")) {
                                                                # if offline user and new
pending subscription
                                                                if
(!target_online("registrar")) {

        #%subscription_status="waiting"; # store it as waiting subscription
                                                                        xlog("L_ERR",
"storing 'pending' winfo to: %tu, from: %fu\n");

        store_winfo("registrar");
                                                                }
                                                        }
                                                }
                                        }
                                        break;
                                }

                                if ((@msg.supported=~"eventlist")) {
                                # such user doesn't exist and Supported header field
                                        #       -> probably RLS subscription

                                        if (lookup_domain("$td","@ruri.host")) {
```

```
                                                        if (lookup_user("From")) {
                                                        if (is_simple_rls_target("$uid-list")) {
                                          # log(1, "it is simple subscription!\n");
                                          # takes From UID and makes XCAP query for user's
                                               # list named "default"
                                                                 if  (!query_resource_list("default"))
{
                                                                 t_reply("404", "No such user list");
                                                                                     break;
                                                                           }
                                                                 }
                                                        }
                                               }

                                               if (!have_flat_list()) {
                                                  # query_resource_list failed or was not called
                                                  # do standard RLS query acording to To/AOR
                                                        if (!query_rls_services()) {
                                                                  log(1, "XCAP query failed\n");
                                                                 t_reply("404", "No such list URI");
                                                                  break;
                                                        }
                                               }

                                               handle_rls_subscription("1");
                                      }
                                      else {
                                               # not resource list subscription -> invalid user
                                               xlog("L_ERR", "subscription to invalid
user %tu\n");
                                               t_reply("404", "User not found");
                                      }

                                      break;
                             }
                             else {
                                      # renewal subscriptions - try to handle it as RLS and if
failed, handle it as PA subscription
                                      # FIXME: better will be test like
existing_rls_subscription()
                                      #              and  existing_subscription("registrar")
                                      if (!handle_rls_subscription("0")) {
                                               handle_subscription("registrar");
                                      }
                                      break;
                             }
                    };

                    if (method=="PUBLISH") {
                             if (!t_newtran()) {
```

```
#                               log(1, "newtran error\n");
                                sl_reply_error();
                                break;
                        };
                        handle_publish("registrar");

                        # deliver messages to online user
                        # TODO: only if user goes from offline to online?
                        if (target_online("registrar")) {
                                # log(1, "Dumping stored messages\n");
                                # dump stored messages - route it through myself
(otherwise routed via DNS!)
                                if (m_dump("sip:127.0.0.1")) {
                                        xlog("L_ERR", "MSILO: offline messages
for %fu dumped\n");
                                }
                        }

                        break;
                };

                if (method=="NOTIFY") {
                        if (!t_newtran()) {
                                log(1, "newtran error\n");
                                sl_reply_error();
                                break;
                        };
                        # handle notification sent in internal subscriptions
(presence_b2b)
                        if (!handle_notify()) {
                                t_reply("481", "Unable to handle notification");
                        }
                        break;
                };

                if (method=="MESSAGE") {

                        if (authorize_message("http://localhost/xcap")) {

                                # use usrloc for delivery
                                if (lookup("location")) {

                                        log(1, "Delivering MESSAGE using usrloc\n");
                                        t_on_failure("1");
                                        if (!t_relay()) {
                                                sl_reply_error();
                                        }

                                        break;
                                }
                                else {
                                        # store messages for offline user
                                        xlog("L_ERR", "MSILO: storing MESSAGE
for %tu\n");

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

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

                                        if (m_store("0", "sip:127.0.0.1")) {
            #                          log(1, "MSILO: offline message stored\n");
                                                if (!t_reply("202", "Accepted")) {
                                                        sl_reply_error();
                                                };
                                        } else {
                                                log(1, "MSILO: error storing offline
message\n");
                                       if (!t_reply("503", "Service Unavailable"))
{
                                                        sl_reply_error();
                                                };
                                        };
                                        break;
                                }
                                break;
                        }
                        else {
                                # log(1, "unauthorized message\n");
                                sl_reply("403", "Forbidden");
                        }
                        break;
                }

                lookup("aliases");
                if (!uri==myself) {
                append_hf("P-hint: outbound alias\r\n");
                route(1);
                break;
                };
```

```
			# native SIP destinations are handled using our USRLOC DB
			if (!lookup("location")) {
				sl_send_reply("404", "Not Found");
				break;
			};
		};
#		append_hf("P-hint: usrloc applied\r\n");
		route(1);
}

route[1]
{
		# send it out now; use stateful forwarding as it works reliably
		# even for UDP2TCP
		if (!t_relay()) {
			sl_reply_error();
		};
}


failure_route[1] {
		# forwarding failed -- check if the request was a MESSAGE
		if (!method=="MESSAGE") { break; };
		log(1, "MSILO: MESSAGE forward failed - storing it\n");

	    # we have changed the R-URI with the contact address, ignore it now
		if (m_store("0", "")) {
			t_reply("202", "Accepted");
		} else {
			log(1, "MSILO: offline message NOT stored\n");
			t_reply("503", "Service Unavailable");
		};
}
```

COS/CCS 2008-24