Context-aware applications for a Pocket PC

YU SUN

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2007

COS/CCS 2007-28

Context-aware applications for a Pocket PC

by

Yu Sun

Submitted in partial fulfillment of
the requirements for the degree of

Master of science (Information Technology)

at the

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

Advisor and examiner: Gerald Q. Maguire Jr.

17 December, 2007

Acknowledgement

After one and a half years' studying for this Master's degree, I learned one thing, that I could never have ***any*** of these accomplishments without the supports and encouragements from a lot of people whom I would like to give my sincerely appreciation.

First and foremost, I would like to express my deepest gratitude to my adviser and examiner: Professor Gerald Q. Maguire. You are not only advising me in an academic way which is the right path for a Master's engineer, but also you are guiding and helping me to build the success for my future career. I can not imagine how much time you spent for commenting my report, which gives me so many valuable feedbacks and advises. Also I can not forget the lectures you gave me for how to be a person that does not afraid of any challenges. If I do take the academic path, I only hope that I am able to do half of what you have done to me.

I would also like to thank Athanasios Karapantelakis as being the technical stuff to help me with my implementation and explaining so many questions that I am not clear of. I felt quite lucky of being in such a good research environment with so many nice people, such as Alisa Devlic.

This thesis is part of the project which consists of three Master thesis. So here I would like to thank my colleagues Mohammad Zarify Eslami and Daniel Hübinette. I can not forget the nice six months we had together, and I can not even think of finishing my part without your help. Also thanks to Haruumi Shiode for his opposition and helpful suggestions.

My friends, who supported me through this period, I would like to thank you, but I am sorry that I can not name you all here. Finally, and as always, this thesis is for you, my beloved parents.

Abstract

With the rapid development of technology for context awareness, pervasive computing is releasing people from their traditional desktops. Since mobile devices feature portability and are (nearly) always connected, people tend to carry them wherever they go. Hence, devices such as cellular phones and Pocket PCs are the most suitable platforms for developing context aware applications which users will utilize in their daily life. For these context aware systems, using this context information not only improves the user experience of ubiquitous computing, but also lets the system know who you are or what you have. More importantly, the device can know where you are and predict what you might like to do, thus simplifying many of the user's interactions with devices and other people around them.

This thesis project involves the design, implementation and evaluation of a context aware application, based upon a Pocket PC, that can remind the user of tasks when the user approaches the relevant location for this task. The application interacts with a context aware infrastructure by using the SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) protocol, receives context information for the user described using XML. A number of new tags, based upon a new XML schema, have been introduced for this task.

This context aware mechanism enables the user to receive any form of information updated by the context server. In this thesis, updates to this information are driven by changes in the user's location. Additionally, by using the existing calendar application on the Pocket PC, the user can experience location based reminders *without* learning how to use a new user interface.

Sammanfattning

Med den snabba utvecklingen av kontextmedvetna teknologier befriar den genomträngande datoriseringen människor från deras traditionella datorer. Eftersom mobila apparater medför bärbarhet och är (nästan) alltid uppkopplade, tenderar människor att bära dem överallt. Följaktligen blir apparater som mobiltelefoner och Pocket-PC de mest passande plattformarna för utvecklandet avkontextmedvetna applikationer för daglig användning. För dessa kontextmedvetna system kommer inte bara användandet av kontexinformation förbättra användarens upplevelse av överallt förekommande datorisering, utan låter även systemet veta vem du är eller vad du har. Ännu viktigare är att apparaten kan veta var du befinner dig samt förutsäga vad du skulle kunna vilja göra, och därigenom förenkla mycket av användarens interaktion med andra apparater och människor i omgivningen.

Detta examensarbetsprojekt involverar designen, implementationen och evalueringen av en kontextmedvetet applikation, baserad på en Pocket-PC, som kan påminna användaren om uppgifter när användaren närmar sig det relevanta området för dessa uppgifter. Applikationen interagerar med en kontextmedveten infrastruktur genom användandet av protokollet “SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)”, mottas kontextinformation för användaren beskriven i XML-format. Ett antal nya taggar, baserade på en ny XML-schema, har introducerats för denna uppgift.

Denna kontextmedvetna mekanism gör det möjligt för användaren att ta emot alla typer av uppdaterad information från kontextservern. I denna avhandling uppdateras denna information genom att användaren förflyttar sig. Dessutom kan användaren, genom att använda den befintliga kalenderapplikationen i Pocket-PC:n, få lägesbaserade påminnelser skickade till sig utan att behöva lära sig använda ännu ett interface.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today, more and more people are using mobile devices to substitute for their traditional alarm clock. That is because most mobile devices support text input along with alarms, hence the user can set a message to appear reminding them what to do at a specified time. When this specified time comes, the mobile device will generate an alarm and the user could see the text that he/she has written earlier (or that might have been generated by some service). However, this approach has some flaws. Since there are many events that are based more upon location than time, using time as the only context trigger is not suitable for many situations (or more specifically location dependent events). However, using location as an additional element of context information together with time could enable the reminder service to generate an alarm at much more appropriate times and locations.

There are two main problems when developing such an application. First is location detection. How should the information concerning the detection of location be transmitted to the mobile device (in our case this device is assumed to be a Pocket PC), and which protocol should we use to communicate with the mobile client from the rest of the system?

Location is one of the most important factors in people's daily activities, and there are quite a few location-based applications on the market or under development. The most difficult issue when developing a location-based application is getting the device's location. Currently the most popular means of doing so is using the Global Positioning System (GPS)[1], others are sensing the location by collecting data from different sensors, or using positioning technologies with the Global System for Mobile communications (GSM), Wireless-LAN (WLAN)/Wi-Fi networks, etc.[2][3][4][5][6][7].

Another problem is: When designing and implementing a mobile application, not only

do functionality in the mobile device need to be considered, but the convenience which the application can bring to the user is also crucial. Therefore, the user should not be bothered with re-configuration simply because he/she shifts to using another mobile device, as the user will still want the same functionality that the application regularly provides. This requires the application to be compatible with all kinds of devices.

1.1 Concept of context awareness

Dey and Abowd[8] describe context as:
"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 application themselves."

As Jen-yi Pan writes in [9], human attention is a scarce resource. This means when realizing pervasive computing, we should minimize the user's distractions due to system complexity and make the interaction between human and system as simple as possible. For example, assume a user has several mobile computing devices with him, when the user moves to another location, he would be unwilling to interact with each of these devices and manually tell the system that the surrounding context for each device has changed. Instead, the system should be able to collect this information itself and make decisions for the user automatically. Therefore, context awareness is needed for such a system. Additionally, making a successful & friendly interface between the user and the system is also an important requirement for context aware systems.

Following from what A.K. Dey and G.D. Abowd stated in the quote above, when developing an application, we should think about additional context information which we can use. For example, when authenticating a person, in many cases not only can be a password used, but additional means of identifying this person can be used to increase the probability of correctly identifying the person and avoiding incorrectly misidentifying them. It is the additional context information and its use in an application which is the focus of this thesis project.

1.2 Problem statement

Time and location are the most common context information. The first requires less system complexity since an approximate time can be provided by a simple clock and more precise time using a more sophisticated clock, network time protocol, etc. But, using location as context information requires that the system has some mechanism to support positioning or sensors for detection of the device's location.

It used to be difficult to acquire the location of a mobile device. However, with the development of positioning and sensing technology, adding a location aware mechanism to an alarm device is feasible and could be valuable to the user (and potentially others). The idea of a location aware reminder comes from actual scenarios in which people tend to forget things after they are reminded.

Today's reminder systems mostly use time as the only context information. After the user has written a textual message, the only change in the system state which is considered is time. Below we give an example in which people tend to forget events after/before they are notified (It has already happened to me!).

> *I borrowed a book from the library and it is now due to be returned, I carried it to school and planned to return it. However, when I passed by the library, I forgot that I had this book and I missed the opportunity to easily return it. Although I have set my alarm to notify me at the time when I expected to be in the building where the library is, the problem is that the device did not know when I was passing by the library. So I was notified when I was already in class; then my reaction was to turn off this alarm, unfortunately I might not think of returning the book again until I return home and found the book still in my bag!*

Therefore, as location of the user can be detected more easily today, we should exploit this context information to improve the accuracy of presenting an alert to the user in order to meet the user's expectations.

1.3 Organization of the thesis

This thesis will investigate, design, implement, and evaluate a system where a user instead of needing to learn a new application can use the built-in calendar on the Pocket PC as the

user interface to a location-based reminder service.

The remainder of this report is organized as follows:

Chapter 2 investigates the existing relevant products which provide the same or similar functionality of the location base reminder on a Pocket PC. Chapter 3 introduces a generic standard for the event notifications and several context aware architectures. Chapter 4 states the goals of this thesis and describes the implementation. Chapter 5 describes the evaluation methodology and presents the results of an evaluation of the implementation. Finally, Chapter 6 summarizes the evaluation data, concludes the report, specifying some of its limitations and future work.

Chapter 2

Background

2.1 Existing Location-based reminders

Location based reminders are often classified based upon the positioning technology that has been used in the mobile device. Most location systems use one of the following techniques or sometimes a combination of them[10]:

Triangulation: Using multiple distance measurements between known points (lateration), or via angulation which measures angle or bearing relative to points with known separation.

Proximity: Determine if one of a known set of locations is nearby, commonly done using one of three means:

- Detecting physical contact or near contact.
- Monitoring wireless cellular base stations (which transmit their ID)
- Observing automatic ID systems.

Scene analysis: Examining a view from a particular vantage point. Comparing a set of known views (of known locations) with the current view captured by either a wearable camera, or when the scene is any measurable physical phenomena other than a visual image, with the observed features.

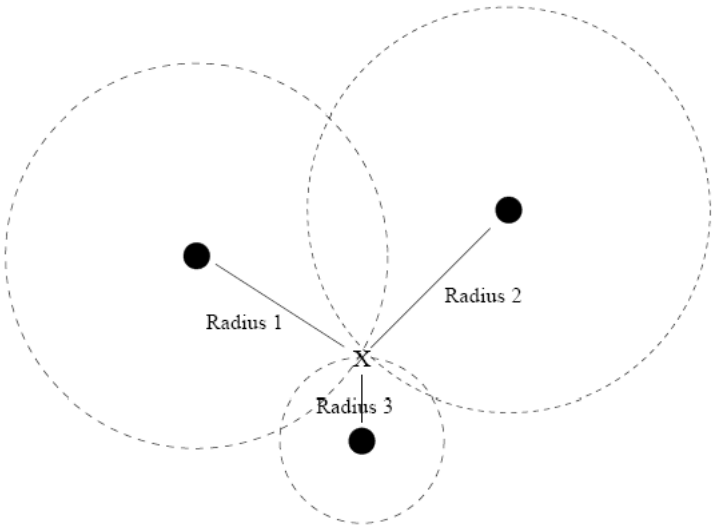


Figure 2.1: Determining 2D position using literation. The position X is determined by the distances between the 3 non-collinear points.

As previously mentioned, many of the existing location-based reminders detect the user's current location using GPS[11][12], GSM[4][13], and WLAN[2] or Wi-Fi[14] networks. GPS is most commonly used in an outdoor environment since it is widely available around the earth using signals from orbiting GPS satellites. However, when the user needs to tell the difference between two nearby rooms in the same building or two rooms on different floors of the same building, although the GPS system can provide three dimensional location (latitude, longitude, and altitude) plus the time, there is a problem, since the reception of the GPS signals is limited within buildings due to radio attenuation by the building materials[5], hence the GPS system is not generally useful indoors. The GSM network has significant indoor and outdoor coverage, but the precision of positioning is much worse than GPS, although the result from a survey in[15] suggests that users are more concerned about coverage than accuracy when using location based reminders, there were still quite a few complaints about the inaccuracy of GSM based location systems due to the large cells of a typical GSM network[16].

Therefore, we would like a positioning system for an indoor environment that had a precision comparable to the GPS system outdoors. Today, WLAN, Bluetooth, and other ad-hoc networks are the most common means of sensing location indoors. These systems generally work by either triangulation or detecting the existence of a Bluetooth device nearby (i.e., proximity)[5]. For identifying an object that the system is to keep track of, the first and most widely used approach is Radio-frequency identification (RFID) tags. These RFID tags are usually attached to the objects[14] such as a person's mobile phone. The system utilizes

these RFID tags to both sense the location of the device and identify the user's identity when he or she enters a specific place. As every Bluetooth device has a 48 bit media access and control (MAC) layer address we can utilize this address as a virtual Bluetooth tag to identify the user as well[5] (this assumes that the Bluetooth interface is turned on).

2.1.1 Place Mail

Developed by Pam Ludford, Place Mail[11] is a location-sensitive to-do list appliance built upon GPS-enabled cell phones. It stores the latitude and longitude of the user's favorite spots into the cellular phone's memory. The user can write notes or record voice notes corresponding to these spots. When the user passes within a half-mile ($\cong$ 1 km) of the spot that is on the to-do list, the cellular phone beeps and delivers the customized text message or verbal reminder to the user. In the examples given in [11], the user gets "Get Crouching Tiger, Hidden Dragon" or "Buy rope caulk" from the cell phone when they pass by the specified places.

The mobile device used in Place Mail is the Motorola I88S cellphone. A key advantage of this model is that its GPS antenna is better than the antennas in typical flip-open cellular phones. This enables the device to receive GPS signals even when it is in a purse or covered by thick clothes.

2.1.2 ComMotion

The MIT Media Laboratory developed a location-based context-aware system called ComMotion[12] in 2000. The system, utilizes GPS technology for location-sensing, maintains a to-do-list for the user's selected locations, with each item consisting of both text and voice recordings. Also, it records the times that the user has passed by these specific locations. When a location is frequently detected, ComMotion will display the location on a map and prompt the user to name it. The user can either name such places (as bank, home, school, etc.) or tell the system to ignore this location (when the user believes that this location may never be used). The hardware of ComMotion includes a portable PC, a GPS receiver, a Cellular Digital Packet Data (CDPD) modem, and a Jabra earphone with a bone conduction microphone.

The system consists of a server and a client for human-computer interaction. The client side of the human-computer interface has both speech and graphical user interfaces. The server side of the system is located elsewhere on the Internet. These servers store maps,

the messages each user has input, and other information that is relevant. The client communicates with all of the different server processes via TCP/IP sockets. Figure 2.2 illustrates the three main modules of the client application and its connection to the server in ComMotion.

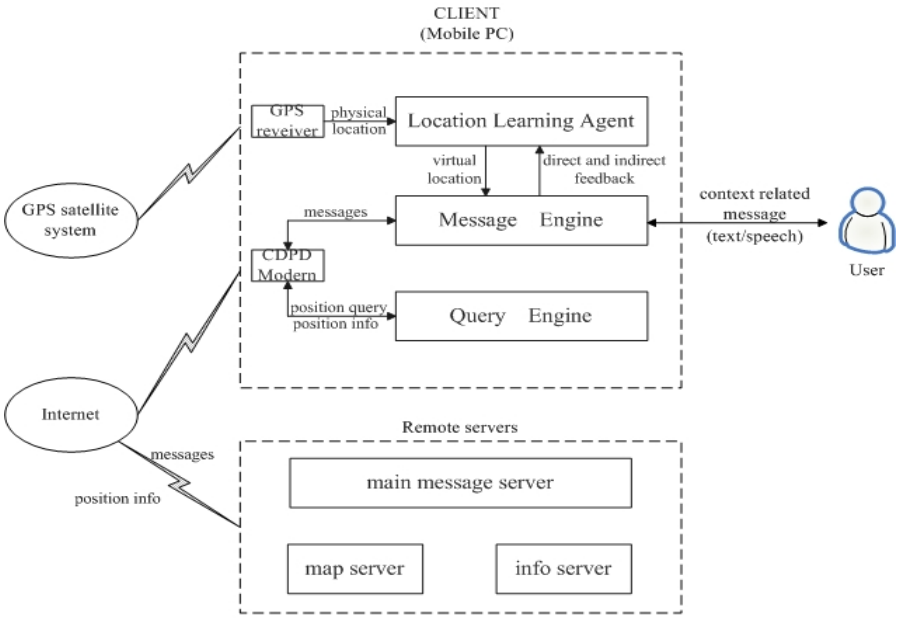


Figure 2.2: The architecture of comMotion showing the three main modules of the client application and its connection to the server (Adapted from [12])

2.1.3 Geominder & Place-Its

Geominder[13] is one of the applications developed by Ludimate[1] The main feature of Geominder is that it allows the user to create location-based reminders that are attached to physical locations. This appliance utilizes the cell tower ID of the GSM network[17]. When the user is in a specific place and creates a reminder corresponding to that location, the Geominder retrieves the cell tower ID from the local GSM network and stores it as the location identifier. Compared to Place Mail, the improvement Geominder brings is that it uses a symbolic location instead of the physical position, that is, instead of storing latitude and longitude of specific spots, it symbolizes the locations as icons that are much easier for the user to remember.

As illustrated in Figure 2.3, the user sets a voice reminder corresponding to the market and gets a verbal reminder when he or she approaches the market. Additionally, the user can

[1]Ludimate is a company that creates games for mobile phones.http://ludimate.com/index.php

Figure 2.3: Voice nodes setting for Geominder[13] (These figures appears here with permission from Jorge Diogo, Ludimate Support.)

add more locations to Geominder, for example, when he is in the school he can add the school icon causing Geominder learn the cell ID to be associated with this "school".

Geominder is an application that runs on Symbian[2] devices, compatible with Symbian Series 60 smartphone devices such as Nokia 3230, 3600, 3620, 3650, 3660, 6260, 6600, 6620, 6630, 6670, 6680, 6681, 6682, 7610, 7650, N-Gage, N Gage QD, N70, and N90.

Although it is considered a better application than systems using GPS to sense their location (such as Place Mail) Geominder still has some weaknesses. One of the biggest problems is that the size of cells in the GSM network varies from place to place. For example, the cells in locations that have low population density are typically much larger than those in a downtown area. Therefore the user could be reminded even when he or she is miles away from the expected spot. On the other hand, since even the smallest cell is bigger than a building, when we want to use cell IDs to distinguish rooms, Geominder no longer functions as expected.

Place-Its[4] has almost the same features as Geominder. It only differs in the graphical user interfaces.

2.1.4 Location Based Reminder

Pezhman Givy[18] developed the Location Based Reminder (LBR) as a location based reminder for Symbian OS S80/2.0 mobile phones such as Nokia's models 9300(i)/9500. Similar to Geominder, it also uses the GSM cell ID to define the location. LBR supports

[2]Symbian OS - the mobile operating system.http://www.symbian.com/

two types of actions for each reminder: SMS and a reminder box. The user has the option to set the LBR to remind him either when he enters a location or leaves.

As an application, LBR optionally supports GPS devices to find where the cell is located. That is, when you attach a GPS receiver to the phone, LBR annotates each cell with its GPS coordinates. Therefore it can also be used as an application for collecting the location of GSM cells.

2.2 Related products

2.2.1 Gate reminder

In 2004, a group of people from Samsung[3] invented an context aware reminder called the Gate Reminder[14]. The main idea behind the Gate Reminder is that when the user is going to leave their house as they approach the front door, they will be reminded with either a text message or a list of missing objects that he or she has registered but does not have with them.

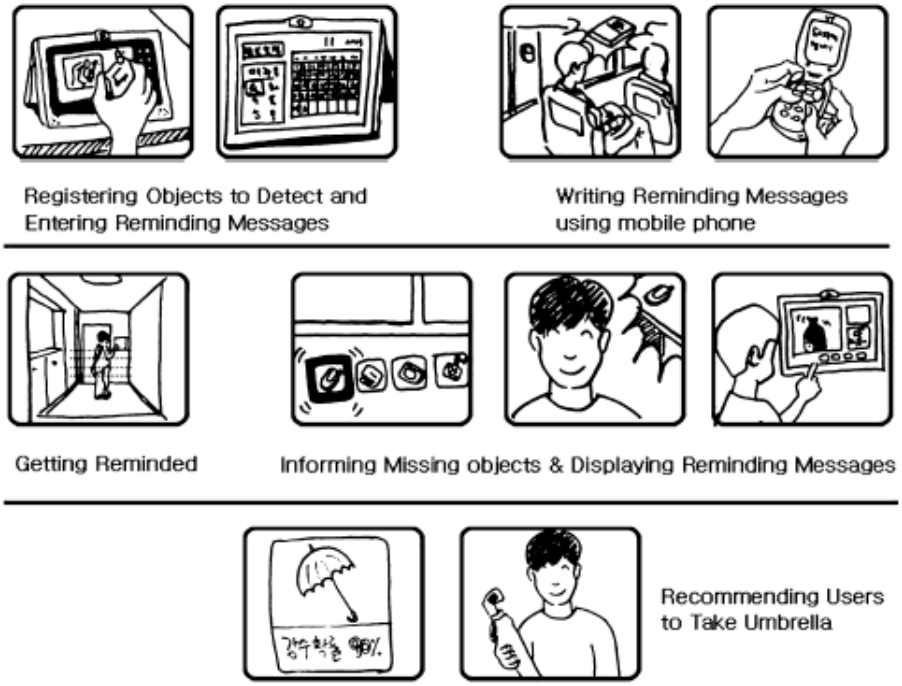


Figure 2.4: Use scenario of the Gate Reminder. (These figures appear here with permission from Sanghyun Park, the author of [14].)

[3]Samsung.http://www.samsung.com/

To enable the system to know who is approaching the door, each user should have a unique RFID tag with him/her[19]. These RFID tags are also attached to the user's belongings such as cell phone, keys, wallet, etc. in order to act as object IDs. The user registers these objects with the corresponding RFID tag with their Gate Reminder, hence when he or she walks through the front door, the Gate Reminder will first detect the user's identity, then it retrieves the information that the user has input, displaying messages if there are any messages for this user. If any of the registered objects are missing, for instance the wallet that the user has forgotten to take with them, the screen on the door will display the icon of a wallet to remind the user that they have possibly forgotten this object. Also, the Gate Reminder can display the weather on that day and make recommendations to the user such as to take their umbrella. Figure 2.4 shows the use scenario of the Gate Reminder.

2.2.2 ActiveCampus

ActiveCampus Explorer[2] is an application using another location-based technology. It is a typical PDA based context aware application. The information provided to the user is adjusted based upon the user's location. For example, when the user enters a specific location, there will be a map of the user's vicinity displayed on the screen. The user can utilize this information in order to perform further actions.

The location sensing is done by computing a signature based upon the set of 802.11b (WLAN) access points which can be heard currently[20]. Users' point-and-click to make corrections to the map locations, these are saved along with the reports of WLAN signal strength, enabling the system to refine future location inferences[2].

Also, ActiveCampus Explorer provides services and information based on different locations. An example is shown in Figure 2.5. When the user starts this application, he sees a map of his vicinity that is provided by the nearest web server[21]. On the map there are many overlaid labels that the user can click on, each is connected to a URL. In this case, the user chooses Buddies, then the system displays the presence information of each person in his buddy list.

In [2], ActiveCampus uses HTML to achieve instant updating. At the server side, it refreshes the web page periodically so that the user gets the updated message on time. But for a mobile device, refreshing a web page is relatively slow compared to a device with a wired connection. Moreover, the mobile device does not have unlimited power to support constant refreshing, hence the authors mention at the end of [2] that currently they are trying to use

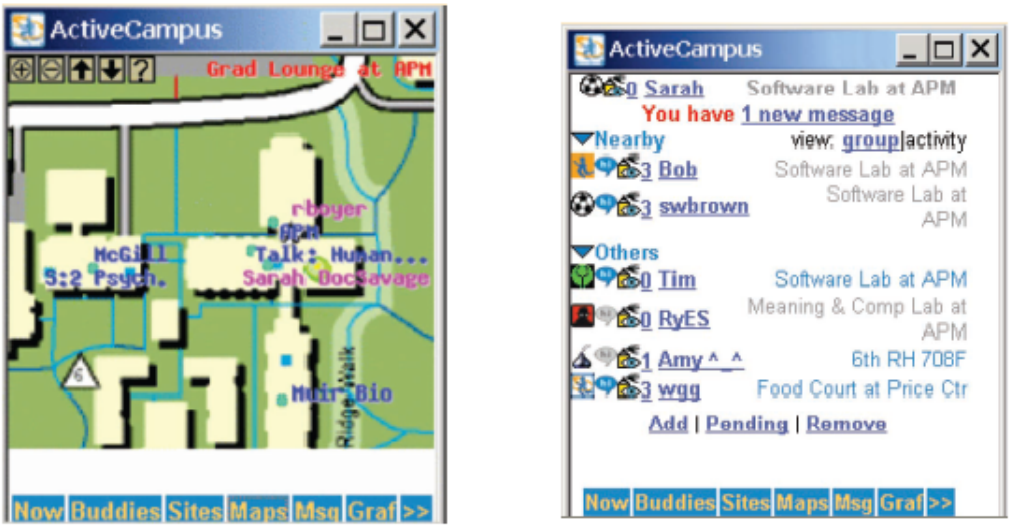


Figure 2.5: Map (left) and buddies (right) pages of ActiveCampus, shown on an HP 548 Jornada for a user "Sarah." Outdoor or indoor maps of the user's vicinity are overlaid with buddies, sites, and activity links. (These figures appear here with permission from William Griswold, the author of [2].)

XML-based instant-messaging frameworks to utilize a push connection from the server to the client, so that users can get more immediate alerts.

2.2.3 Geonotes

The design idea behind GeoNotes[22][23] is that the information that is usually available in augmented reality (AR) systems[24] is mostly created by professional content providers, and such information are often a bit "clumsy" for the user. This occurs because the AR information spaces lack dynamic, social context and does not exploit communication between users and the system.

Therefore, GeoNotes was designed to enable users to place virtual annotations in geographical spaces via mobile devices. It detects the user's current geographical position and allows them to write notes and graffiti at that place automatically. Also, the user can read, browse, or search the GeoNotes of other users. Additionally, GeoNotes has extensive functionality. It enables the user to write notes either anonymously or with their signatures; when receiving the GeoNotes, the user can set filters according to their preferences. For example, they can simply reject spam-GeoNotes or selectively view GeoNotes based upon their authors.

The GeoNotes system consists of a server, a client, and four databases. The client is a JAVA application that uses the Simple Object Access Protocol (SOAP)[26] which carries XML encoded remote method invocations over HTTP to communicate with the server. The

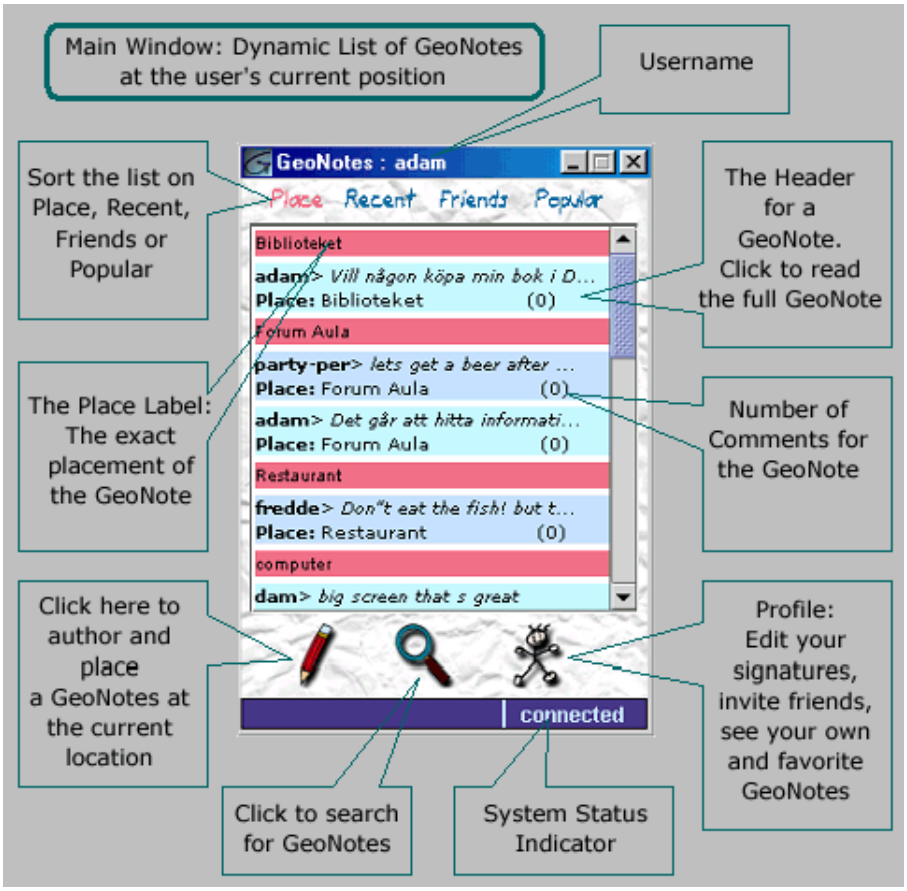


Figure 2.6: The main window of use scenario of GeoNodes (This figure appear here with permission from Fredrik Espinoza, the author of [25].)

system detects the client's MAC address when the client connects to a new access point. Because the latitude and longitude of each access point is stored in a MySQL database[25], the system then knows the approximate geographical location of the client.

Chapter 3

A generic standard for event notification

3.1 Event notification in a location-based reminder

When the user's location has been detected by the system, usually there are two tasks the context aware architecture has to perform:

- After learning of the user's presence, the system retrieves the data relevant to this user.
- This information has to be sent to the user. For example, this information might include the user's location and information related to the user's presence at this location.

In section 2.1 we discussed the detection of the user's presence, therefore here we focus on the later task. A location based reminder, as an application on mobile devices, should be able to receive a notification about its location from the context aware system when it enters a specific place. Hence, it must either parse the collected data from the environment (from various sensors, where some of these sensors could also be virtual) directly or receive this notification from a server which sends only the relevant information.

3.2 Architecture of context aware models

Early context aware models were simple and straightforward, each system was built on a specific platform and interacted with a specific set of sensors. The Active Badge[27] is generally considered one of the first context aware platforms. This system's architecture was specifically designed for infrared sensors. This architecture is easy to understand; however, it has some disadvantages. First, as the devices and sensors are predetermined, such an architecture is difficult to scale and combine with other systems. Second, once designed, this architecture can not be reused if the user wants to deploy another context aware system. Most of the existing products for location based reminder feature such an architecture.

3.2.1 Middleware design for context-aware computing

Today context aware systems utilize a middleware based design which uses middleware to interconnect the sensors and applications. According to Wikipedia, middleware is software that connects software components to support complex or distributed applications. Many designs for middleware for context aware computing have been done. In [28], Henricksen, et al. present a generic layered structure of components for a context aware system. In the layer structure shown in Figure 3.1, a general context aware system consists of 5 layers.

In the bottom layer of this structure there are various actuators and sensors, these sensors could be either hardware sensors or software sensors. An example of the later could be software that records the times of strokes on the keyboard. Layer 1 abstracts the data collected by the sensors into suitably formatted data to be used by other layers. For instance, it might translate the RFID tag ID to a person's name (as a string). Then these context processing components send the information up to layer 2 which has context repositories that provide persistent storage of context information. Finally, decision support tools in layer 3 combine the data from different kinds of sensors and analyzes it in high level situations, e.g. if there are 5 people in the same room, the probability that they are having a meeting is higher than when there are 2 people in the room, therefore all the calls to a person in the room should be transferred to his or her voice mail box. Note that such a room occupancy detector is the subject of a thesis by Daniel Hübinette[29].

Referring to figure 3.1, the middleware for context-aware systems resides between the layer 4 applications and the layer 0 sensors. The main advantage of using such middleware is that it enables the system to be reused when different services are needed, instead of needing to design a new platform from scratch. Also, this model can offer security policy management,

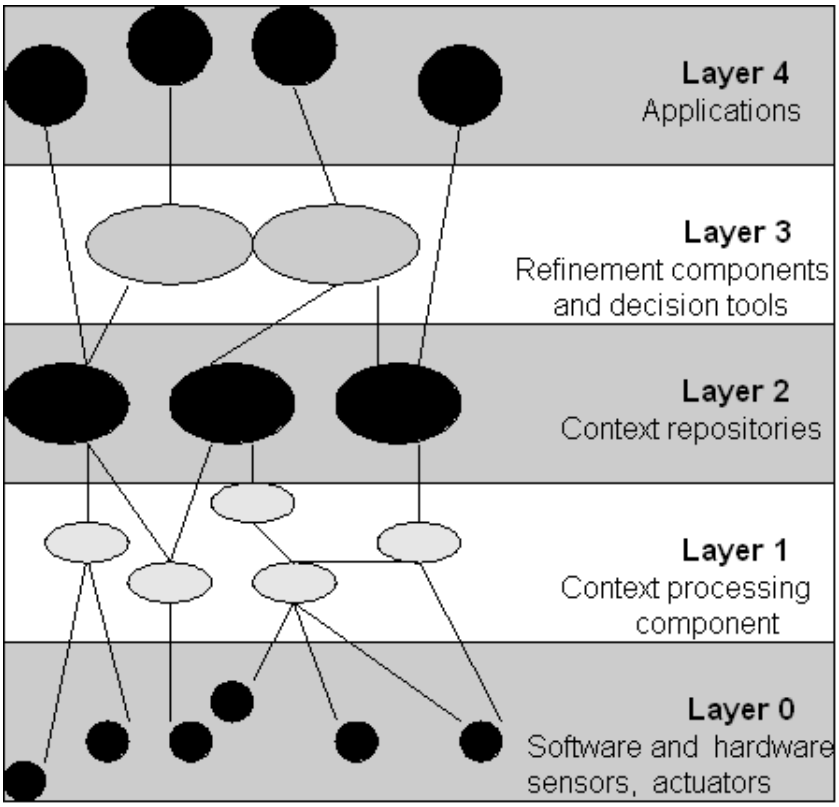


Figure 3.1: A generic layering of the components of a context system

which allows the user (or others) to access only certain context data.

3.2.2 Context Toolkit - A classic middleware based design

Dey et al. implemented a Context Toolkit[30][31] in 1999. For abstracting the sensor data, they introduced a Widget to mediate between a user and the environment. A Server aggregates the context while an Interpreter is responsible for transforming context information by increasing its level of abstraction.

Later in [32], they re-defined the components of the framework as *widgets*, *interpreters*, *aggregators*, *services*, and *discoverers*. These *interpreters*, *aggregators*, and *services* reside between the actuators and *widget*. Together they perform some action after acquiring context from the environment on behalf of applications (e.g. turning on/off the light, displaying the message on a screen). *Discoverers* take responsibility for determining whether these components are available or not, as they mainly perform a look up, driven by the applications, in order to locate suitable *widgets*, *interpreters*, *aggregators*, and *services*.

The Context Toolkit realized all the functions between layer 0 and layer 4 as shown in

Figure 3.1, hence it acts as middleware. Figure 3.2 illustrates the relationship between these components.

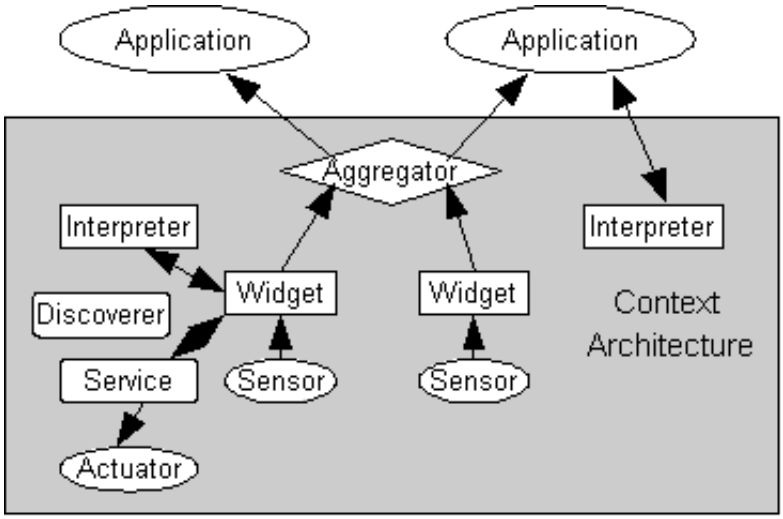


Figure 3.2: Relationship between components of Context Toolkit (Adapted from [32])

The context toolkit implements the above abstractions as Java objects. For communication between these objects, it utilizes a simple object communication mechanism based on HyperText Transfer Protocol (HTTP) and eXtensible Markup Language (XML)[33] encoding of messages. The advantage of using such standard protocols is that it allows components programmed in other languages to interconnect, hence the context toolkit supports heterogeneity.

3.2.3 Other designs for context aware systems

Apart from the original straightforward design of context aware system and the middleware design, H. Chen et al.[34] proposed the idea of Context Broker Architecture (CoBrA). A central server called the context broker which locates in the middle of the architecture, serves as the central processing unit of the context aware environment. It maintains a model of the current context that can be shared by all the devices, services and agents in the same smart space as shown in Figure 3.3.

The CoBrA uses Semantic Web languages such as Resource Description Framework and the Web Ontology Language OWL[35] for defining the ontology of context, instead of in [32] context are implemented as Java class objects which is quite informal. Also, CoBrA provides a common policy language[36] that allows user to control their context information which can enhance the user's privacy. But as a central control unit, the context broker can

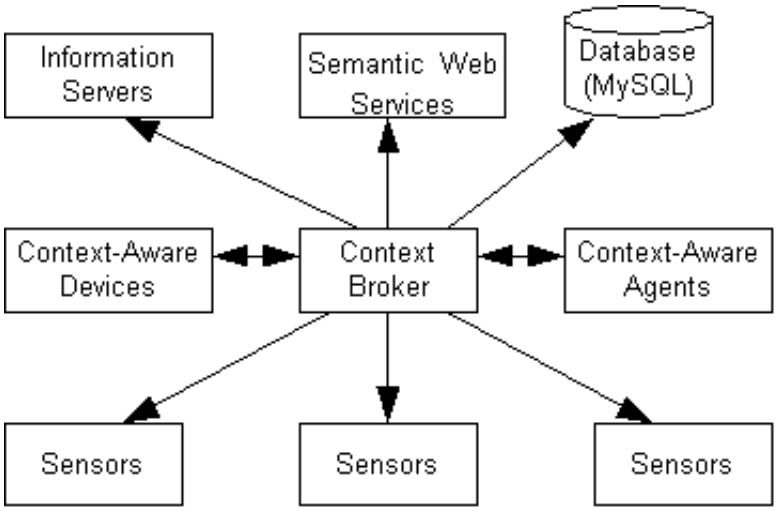


Figure 3.3: A context broker allows concurrent multiple access to sensory data. The CoBrA[34] uses ontologies for manipulating, reasoning, and storing the context information.

be very complicated and difficult to be implemented.

The ACAS project[37] utilizes such a design in a distributed environment, it provides an adaptive, user-centric Internet service to users who move within a heterogeneous infrastructure. It introduced a *Public Service Infrastructure* (PSI) which has a central broker collecting data from a set of sensors, and the context information in this infrastructure can be accessed by the user through a *personal context system* which dynamically connects to the PSI. The context information is exchanged at the application level, thus context acquisition relies only on the user's devices.

Table 3.1: Architectures for context aware systems

Architecture	**Features**
Sensors - Applications	The design is straightforward; usually all the computations take place on the device; once designed, the architecture is difficult to be reused.
Middleware architecture	By using a layered architecture, the applications can access to different sensors from the same architecture. While the middleware provides the context to the applications and does part of the decision making computations for a user.
Context broker	Similar to the middleware design, but the context broker conducts all the computations for a user's decision making. The disadvantage is the complexity of design due to the centralized approach.

3.3 Context models

For describing, transmitting, and storing the context, there are several models to represent the context. Among them, the most commonly used means is the Markup Scheme Model[38]. This scheme was derived from XML, due to the features of XML such as being directly usable over the Internet, it supports a wide variety of applications, it is easy to write programs which process XML documents, it is formal and concise, etc. When considering future development of a context aware application, although the context aware reminder currently uses only location and time as context, it is better to utilize XML instead of using the simple Key-Value models[38], as using XML allows new context information to be easily utilized. It should be noted, that XML only provides a syntax for a hierarchically structured document, it does not define any semantics. Therefore we should choose the most suitable model for making statements about the context.

3.4 Using Session Initiation Protocol

The Session Initiation Protocol (SIP)[39] is a popular standard for communicating over the Internet. SIP has been used for years and there are a lot of SIP stacks, hence it is very easy to find a suitable stack to use when developing applications.

3.4.1 SIP overview

SIP is an application-layer control protocol, it was designed to create, modify, and terminate sessions with one or more participants. It is a peer to peer protocol based upon request-response communication. The entities participating in the session are called user agents. A user agent can function either as a User-Agent-Client (UAC), which is a client application that initiates the SIP request; or as a User-Agent-Server (UAS), which is a server application that contacts the user when a SIP request is received and returns a response on behalf of the user[40]. A user agent can be both the UAC and the UAS in the SIP network, but it can only act as either the client or the server per transaction.

Figure 3.4 shows the basic SIP architecture. The components of a SIP network can be grouped into two categories: clients (endpoints) and servers. In the figure, the SIP User Agents and SIP gateway are SIP clients while SIP servers include the SIP proxy server, SIP redirect server, and SIP registrar server.

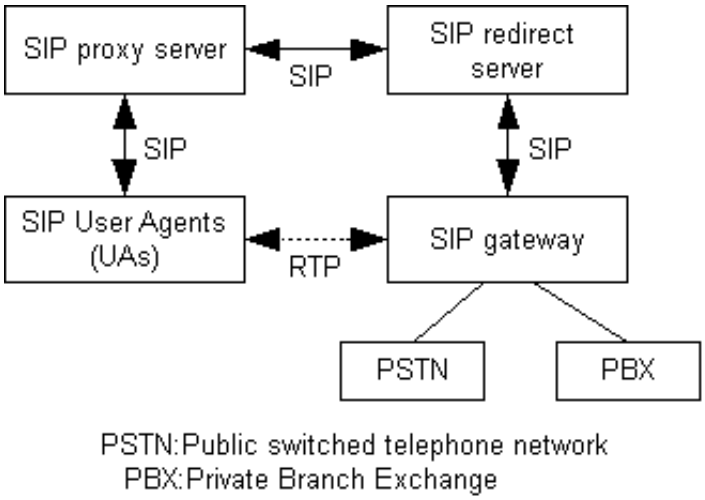


Figure 3.4: SIP architecture

A proxy server receives SIP requests from a client and forwards them on the client's behalf. The redirect server redirects SIP requests, hence it provides the client with information about the next hop or hops that a message should take, subsequently the client contacts the next-hop server or UAS directly. The registrar server processes requests from UACs for registration of their current location; it is often co-located with a redirect or proxy server.

The format of the SIP address is similar to an e-mail address, such as sip:userID@gateway.com. The user ID can be either a user name or an E.164 address (the later is a telephone number with a string of decimal digits). The gateway can be a domain name or an Internet IPv4 or Ipv6 address. Users use their assigned SIP address to register with a registrar server. After registering, this SIP agent can now be contacted by the user's SIP proxy when there is a desire by a UAC to initiate a communication session. Note that a user can have multiple agents which are registered at the same time.

3.4.2 SIP Specific Event Notification

RFC 3265[41] is an extension to SIP. The purpose of this extension is to enable SIP nodes to request notification from remote nodes when certain events have occurred. For this purpose, RFC 3265 introduced two new methods: SUBSCRIBE and NOTIFY. The SUBSCRIBE method is used to request asynchronous notification of an event or set of events at a later time and the NOTIFY method is used to notify a SIP node that an event which has been requested by an earlier SUBSCRIBE method has occurred.

For the SUBSCRIBE request, there should be an "Expires" header which indicates the duration of the subscription[39]. Also, the subscriber should periodically refresh SUBSCRIBE messages using the same "Event" header "id" parameter[1] to maintain the subscription.

For identification of reported events, NOTIFY "Event" headers will contain a single event package name for which a notification is being generated. This name must match the "Event" header in the corresponding SUBSCRIBE message and must include the "id" parameter presented in the SUBSCRIBE message if there is any.

A typical flow of the two new messages described in [41] is:

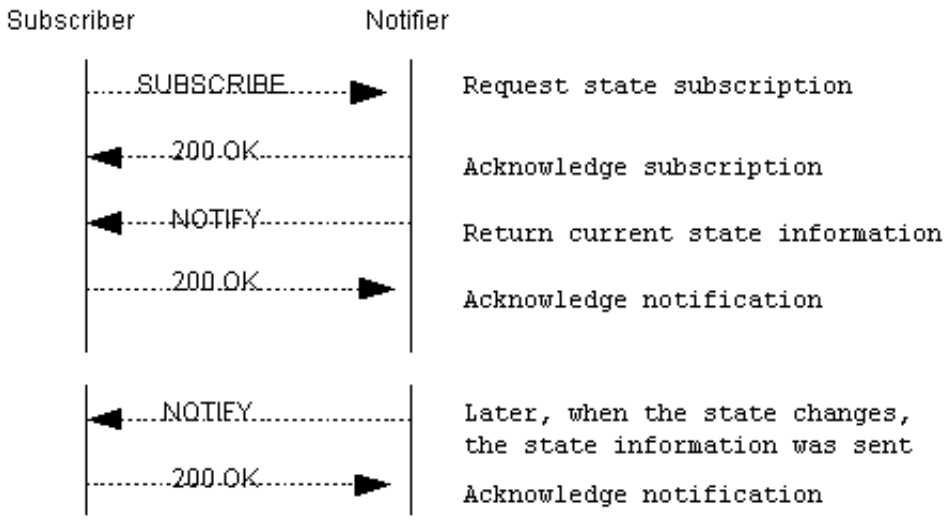


Figure 3.5: A typical message flow of SUBSCRIBE and NOTIFY (Adapted from [41]).

After a SUBSCRIBE request is answered with a 200 class response (indicating success), the notifier must construct and send a NOTIFY message to the subscriber immediately, returning the current state information. Also, when a change in the subscribed state occurs, the notifier should send a new NOTIFY message to the subscriber with the expiration time of the subscription.

[1] If the initial SUBSCRIBE message contained an "id" parameter in the "Event" header, then it is compulsory for the subscription refreshes to contain an identical "id" parameter. Otherwise, it will be considered a new subscription request[41].

3.5 Utilizing presence information

3.5.1 SIMPLE

SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE), is an open standard proposed by IETF. According to [42], the SIMPLE presence specification can be divided into 5 parts: core protocol machinery, presence document, privacy and policy, provisioning, and optimizations.

The core protocol machinery, described in RFC 3265, RFC 3856[43], RFC 4662, and RFC 3903, provides the actual SIP extensions for subscriptions, notifications and publications. RFC 3856 proposes the usage of SIP as a presence protocol, as SIP location services already contain presence information in the form of registrations. Moreover, because the virtual SIP network is capable of routing requests from any user on the network to the server which holds the state of registration, thus SUBSCRIBE requests can be routed to the same server enabling the SIP network be reused to establish global connectivity for presence information[43].

As described in [43], the difference in terminologies from [41] is that a subscriber is now known as a *watcher* and a notifier as a *presentity*. A *Presence User Agent* (PUA) manipulates presence information for a *presentity*, and allows the *Presence Agents* (PAs) to access and manage this information. A PA is a SIP agent which is capable of receiving SUBSCRIBE requests, responding, and generating notifications of changes in presence state which comes from PUAs. Note that both the PUA and PA are logical entities. While a *Presence Server*, is a physical entity that can either act as a PA or a proxy server for SUBSCRIBE requests. An *Edge Presence Server*, is aware of the presence information of the *presentity* since it is co-located with the PUA which manipulates the presence information. [43] also gave a sample message flow, which is shown below in figure 3.6.

The patten of communication is similar to the flow described in [41], only with the PUA which updates presence information added.

3.5.2 Presence Information Data Format (PIDF)

The presence information sent by PUA is passed through presence servers to the subscriber. This information is in the form of an XML presence document utilizing the Presence Information Data Format (PIDF)[44].

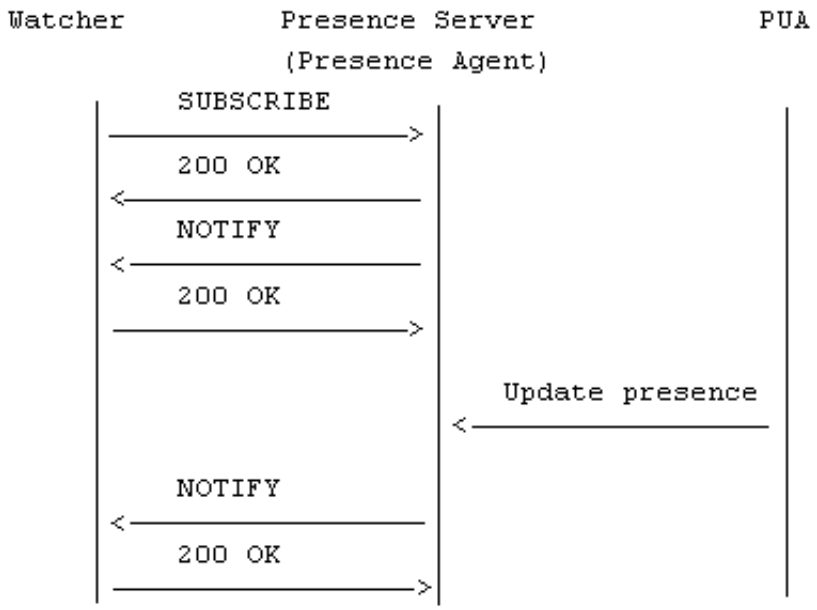


Figure 3.6: Example message flow of of presence update (Adapted from [43]).

A PIDF object is a well formed XML document, therefore it must have an XML declaration and optionally contain a declaration for the encoding method. There are several fundamental PIDF elements defined in [44], all associated with a "namespace". A namespace prefix is in the <presence> element, indicating the namespace in which the presence document is based. After <presence>, there are more elements, which are <tuple>, <status>, <basic>, <contact>, <note>, and <timestamp>.

An example of PIDF with the namespace prefix "Instant Messaging and Presence Protocol (IMPP)" is shown below[44]. The IMPP model document is defined in [45].

```
<?xml version="1.0" encoding="UTF-8"?>
   <impp:presence xmlns:impp="urn:ietf:params:xml:ns:pidf"
       entity="pres:someone@example.com">
     <impp:tuple id="sg89ae">
       <impp:status>
         <impp:basic>open</impp:basic>
       </impp:status>
       <impp:contact priority="0.8">tel:+09012345678</impp:contact>
     </impp:tuple>
   </impp:presence>
```

Note this example does not contain any presence state information updates, since it only defines the basic elements.

The same example using the default XML namespace[44] contains the same information, but has a slightly different look:

```
<?xml version="1.0" encoding="UTF-8"?>
   <presence xmlns="urn:ietf:params:xml:ns:pidf"
       entity="pres:someone@example.com">
     <tuple id="sg89ae">
       <status>
         <basic>open</basic>
       </status>
       <contact priority="0.8">tel:+09012345678</contact>
     </tuple>
   </presence>
```

3.6 PIDF extensions

There are many extensions defined in different RFCs, all with the same goal: offering richer presence information. RFC 4479[46] extended the basic model in [44] by introducing the new concepts of device and person status. In the same spirit, RFC 4480[47] added many more attributes, to allow the indication of activities, moods, places, place types, status-icon, user input, and so forth. RFC 4481[48] extends PIDF by enabling the watcher to know the presentity's **future** plans. It introduced a <time-status> element, which describes the status of a presentity that is either no longer valid or covers some future time period. An example that combines PIDF and timed-status adapted from [48] is shown below:

```
<presence xmlns="urn:ietf:params:xml:ns:pidf"
     xmlns:ts="urn:ietf:params:xml:ns:pidf:timed-status"
       entity="pres:someone@example.com">

     <tuple id="c8dqui">
       <status>
         <basic>open</basic>
       </status>
       <ts:timed-status from="2007-08-15T10:20:00.000-05:00"
          until="2007-08-22T19:30:00.000-05:00">
          <ts:basic>closed</ts:basic>
       </ts:timed-status>
       <contact>sip:someone@example.com</contact>
     </tuple>
     <note>I'll be in Beijing next week</note>
   </presence>
```

Apart from these extensions, in the <status> element arbitrary elements are allowed to appear, but only if they are predefined in the XML Schema and also quoted in the "umbrella namespace" for the <status> element[44].

3.7 Optimizations of PIDF

Presence can be a very expensive service when running over wireless links. There are two main problems: Notifications are often sent when the change is not actually relevant to the watcher. Secondly, each notification contains the full presence state in the notification, rather than just an indication of a change[42].

Therefore, filtering can be specified in subscriptions so that the watchers can limit the cases in which notifications are sent. This was proposed in RFC 4660[49] in conjunction with RFC 4661[50] which specifies the XML format of these filters. To solve the second problem, a partial PIDF document[51] was introduced so that the PIDF document no longer needs to carry all the presence information available for the presentity. This is often important for wireless links, since such links are often low bandwidth, high latency, and/or high cost. Other work has been done to extend XML by introducing a "diff" to represent changes in XML documents[52].

Chapter 4

Goals and Implementation

4.1 Goals of this thesis

4.1.1 Primary goals

The first goal of this master thesis is implementing and evaluating an application that is suitable for handling event notifications for reminders running on a Pocket PC. The application should be able to receive the context updates from the context aware infrastructure, then trigger the alarm of the Pocket PC to remind the user at an appropriate time **and** place.

This context aware application should be evaluated in conjunction with the context aware infrastructure as a whole system. (this includes the presence agent being developed by Mohammad Zarifi Eslami [53], the room occupancy detector being developed by Daniel Hübinette[29], and RFID based location system being developed by Athanasios Karapantelakis). It is import to note that the focus of the evaluation for this thesis will be on the user (both the direct interaction and the indirect interaction, via interaction "behind the scenes" with the application which the user is using - in this case the Calendar application of the Pocket PC). Therefore the details of the **acquisition** of context information will be covered in the other thesis, as will details of the underlying context infrastructure.

4.1.2 Secondary goal

In addition to the location aware calendar application, It is important to consider that this infrastructure should be capable of being extended to include richer context information in the future. Thus extensibility is very important and also needs to be evaluated.

4.2 Implementation methodology

The design and implementation of the location based context aware reminder application on a Pocket PC is divided into several phases. Each of these phases is described briefly below.

First of all, as the reminder application is running on a Pocket PC, there should be an user interface so that the user can input the reminder, indicate at which time the reminder should be presented, and in the case of a location based reminder to indicate where the user wants to be reminded. To achieve this, instead of implementing a new user interface, we instead present a method which takes full advantage of the existing functions already available through applications currently running in the Pocket PC. Thus the user does not have to adapt to a new program, while gaining the benefits of the new service.

In this application, the context information which the user needs to receive from the context aware infrastructure is the user's (current) location. This location information is provided by another application which collects location information, based upon the user having a RFID tag attached to the back side of their Pocket PC. The context aware infrastructure detects the user's presence by collecting information from RFID sensors, processing it, and sending location updates to the user's application. Hence, the application must listen for packets from the context (aware) server. In this case, we use SIMPLE to communicate with the context aware server.

Finally, after received a packet from the context aware server, the application should be able to parse the message and extract the location information for this user. The location information is within XML tags (specifically the tag <location>), which are contained in the PIDF document (which is located after the SIP header). Note that the user must subscribe for location updates and in our implementation only the user's Pocket PC is processing these updates. Thus others can only see that this user is interested in certain location updates, but they need not know where the user actually is. However, to increase privacy all of these packets could be encrypted or sent via a Virtual Private Network (VPN) from the user's SIP proxy (which they already have a trust relationship with). Security and privacy will be discussed further in section 6.6.4.

4.3 Retrieving data from the Pocket PC's Calendar

Rather than implementing a new user interface for a location-based context aware alerting application, we can take advantage of the fact that there is a calendar application already in the Pocket PC. The user can add events by clicking upon a date and time in the calendar. The events are entered as text, along with an optional location, and the time that the user wants to be reminded of this event. In addition, the interface enables the user to turn on/off the reminder, set the sensitivity of the events, etc. It is important to note that a reminder can be for all day - rather than a specific time in the day.

Although it is possible to build a new application with all of the above functions using the .NET framework, we decided to build the application using the built in calendar-appointment application, that is one of the typical pre-installed application in a Pocket PC. The location aware application retrieves data from the built-in calendar when the user inputs information and creates/modifies events in the calendar based upon receiving context information.

The reasons for retrieving data from the built-in application in the Pocket PC, instead of forcing the user to input new information via a new application include: First, since the calendar-appointment application is one of the main functions that the Pocket PC provides, this application should be quite stable and users are used to using this application; Second, when developing a new user interface for a new calendar, it is very hard to correctly size the calendar display adjust well for different devices. For example, for a Pocket PC with a big screen, there seems to be little problem, but if the user has a quite advanced smart watch and he wants to access the same location aware application, the calendar might be difficult to display. However, by using the existing pre-installed application, the screen size has already be accommodated for each specific device. Moreover, in most cases, once the user gets into the habit of using a service, they are reluctant to switch to another, especially if it requires that they learn how to use the new service and its new interface.

Therefore, the best way to develop a location aware application is to do so in such a way that users can continue to use the existing Pocket PC calendar-appointment application. Thus, if the user wants to utilize the location aware service, they simply install & enable the new application on, then this new application will subscribe to a context server in order to automatically retrieve the necessary information based upon their calendar. This application will programmatically manipulate the entries in their calendar in order to realize the desired context aware alerts.

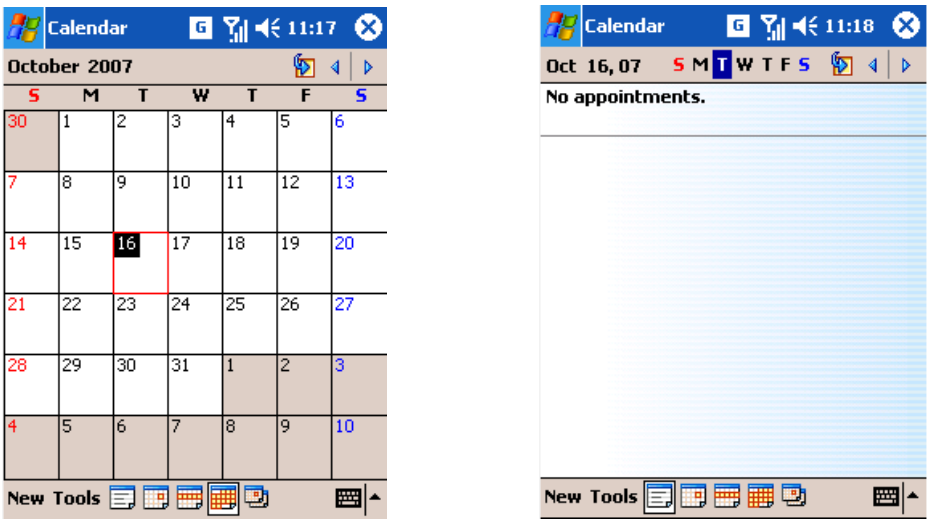


Figure 4.1: The empty calendar before the user adds appointments.

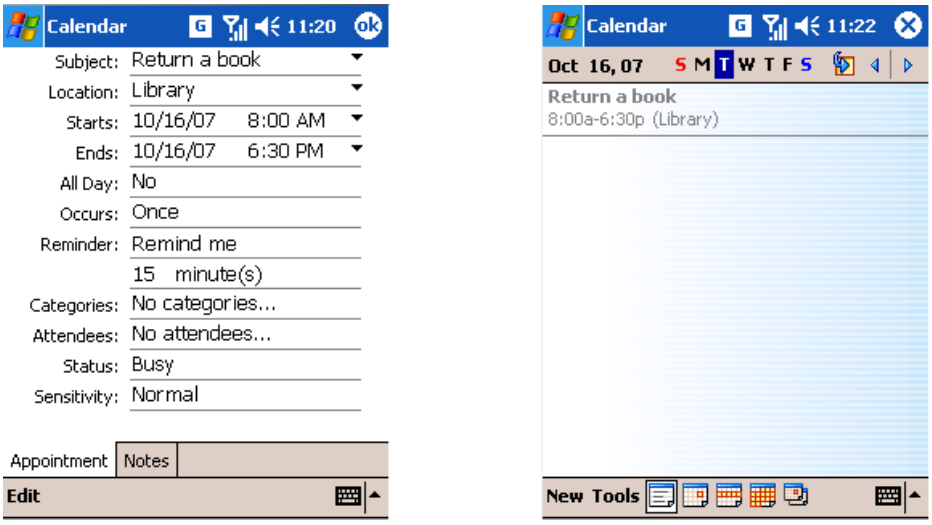


Figure 4.2: The interaction between the calendar and the user, there are many options that the user can set for a single appointment, the right figure shows the appointment added to the calendar.

4.3.1 Pocket Outlook Object Model API

To access the Personal Information Manager (PIM) data on Pocket PCs running Pocket PC 2000 software or later, the Microsoft Developer Network library defines the Pocket Outlook Object Model (POOM) API[54].

When using Compact Framework version 1.0, we have to write numerous PInvoke wrappers and define our own managed object model. However, with the new interoperability feature in Microsoft's Visual Studio 2005 (code named Whidbey) is that it now possible to call Com components directly from managed code, thus it is much easier to use POOM in a managed application in the Pocket PC. By generating an assembly using the Type Library

Importer (tlbimp.exe) SDK tool, you can call POOM directly[55].

The detailed procedures of creating a POOM Type Library can be found in [55]. I used the created file *pimstore.tlb* directly when running *tlbimp.exe* to produce an assembly called *PocketOutlook.dll* which contains a managed code object model that can be used to access POOM directly from a .NET Compact Framework application.

Once a PocketOutlook.dll was created I could reference it from a Visual Studio project and use it just as any other .Net assembly. The result is that I could easily write an application that could both access and manipulate the events in the user's calendar.

4.3.2 Retrieve data already entered in the user's calendar

Consider that a user creates two appointments in their calendar. In figure 4.3, on the left is the event "Buy 10 coupons in the cafeteria", the user chooses "remind" and sets the sensitivity to "Private". The right part of the figure illustrates that the user does not want to be reminded at a specific time, but rather wants to be reminded to get money from the specified ATM machine when he passes it, and in this case he set "Normal" as the sensitivity.

Using the POOM API to write a simple test application, the application can retrieve the information concerning the location, subject of the event, the event's sensitivity, reminder setting, and if set, the date & time. As shown in figure 4.4, the left window is the view of calendar events in the Pocket PC, while the application on the right has successfully retrieved the data from which this earlier user input to the Calendar application. The source code for this simple test application is included as Appendix A.

4.3.3 Retrieving the location element and modifying the alarm time

Taking the advantage of the access to calendar events which POOM brings, the location based context aware application can make use of the location element which the user has previous input. Additionally since it is possible to modify the managed calendar data, when the user enters a specific location and the application learns that the device (assumed to be carried by the user) is at this location, via information from the context infrastructure, then if the time is within the event's valid duration, the application compares this current location to the location which the user defined for this activity in their calendar. If there

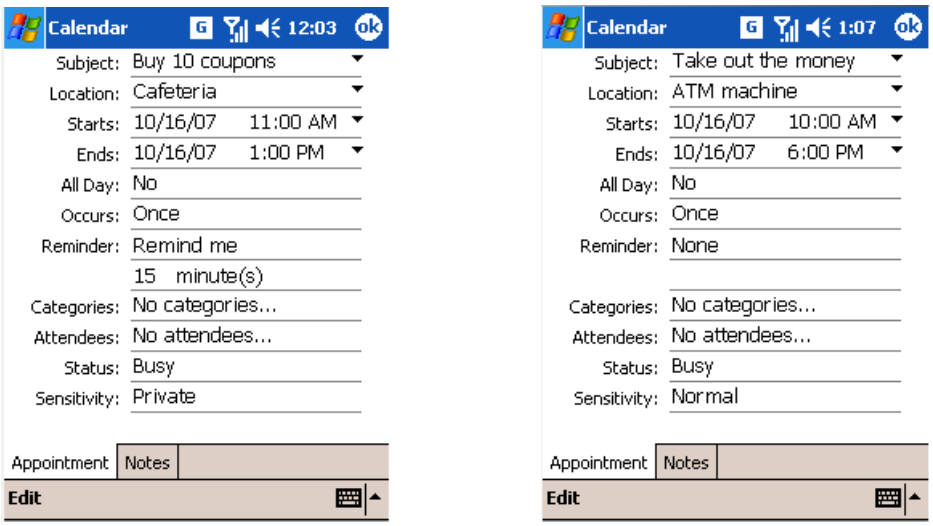


Figure 4.3: Two example appointment from the built-in calendar in Pocket PC.

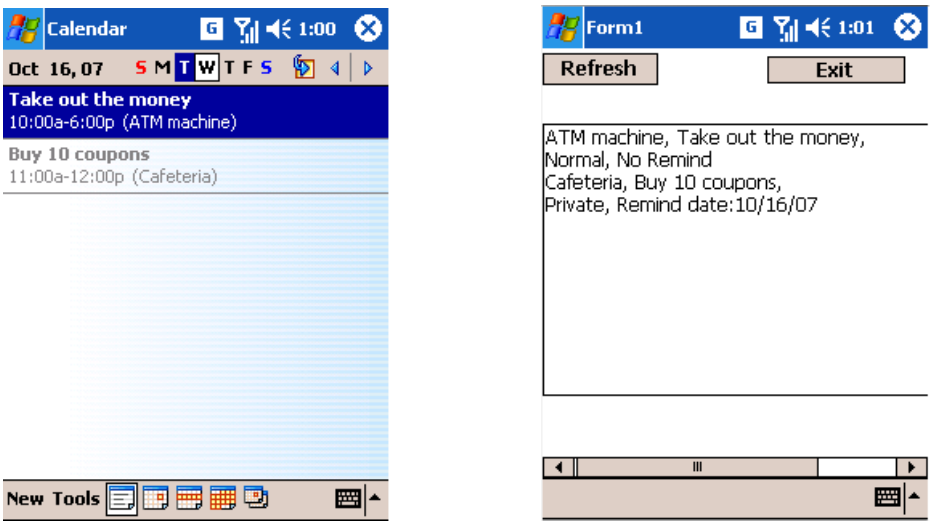


Figure 4.4: Retrieve the data from the built-in calendar in Pocket PC, The left figure shows two items in the user's calendar and the right figure shows that the test application was able to retrieve these events.

is a match, then the application will change the start of the alarm to the current time and keep the end time unchanged. Therefore, the user will get an event notification when he or she enters a specific location. The advantage of this mechanism is that even when the user is not actively looking at this application, by causing an alarm to come from the Pocket PC itself, the user will always receive the alarm.

Programming the above test for the user being in the specified location and updating the reminder time to the current time is as simple as:

```
if ((appointment.Location == location)
&& (appointment.Sensitivity == OlSensitivity.olNormal)){
     if (((appointment.Start <= System.DateTime.Now)
```

```
    && (appointment.End >= System.DateTime.Now))
    ||(((DateTime.Compare(appointment.Start,
    System.DateTime.Today) <= 0)
    &&(DateTime.Compare(appointment.End,
    System.DateTime.Today) >= 0))
    &&(appointment.AllDayEvent))){
        if (appointment.AllDayEvent){
        /* if an AllDayEvent, update the ending time */
           appointment.AllDayEvent = false;
           appointment.End =
           appointment.End.AddHours(23);
           appointment.End =
           appointment.End.AddMinutes(59);
        }
        /* Since the granularity of the time increments which */
        /* the calendar application uses is one minute, so here */
        /* we change the starting time of the appointment to */
        /* the next minute, thus when the current minute expires, */
        /* the alarm of the appointment will be generated */
        appointment.Start =
        System.DateTime.Now.AddMinutes(1);
        /* update the time of the alarm to now */
        appointment.ReminderMinutesBeforeStart = 0;

        appointment.Save();
        /* update the user's calendar */
    }
}
```

4.4 Receiving an event notification from the context aware infrastructure

Following the generic standard for event notification described in section 3, the application will receive the location information from a server which resides within the context aware infrastructure. In order to implement this location based reminder application, we do not need to implement all the SIP functions, since the communication between the application and server is based upon the SIMPLE protocol. Therefore, the application only has to subscribe to the server in order to request the user's location. After the application has received the OK message sent by the server, the application must listen on UDP port 5060 (which is the default SIP port) for event notifications. In the current implementation, the application only accepts notification packets from a specific server. When the SUBSCRIBE expires, the application will send a new SUBSCRIBE message to the server. The format of the SUBSCRIBE message is illustrated below:

```
SUBSCRIBE sip:ccsleft@130.237.15.238 SIP/2.0
Via: SIP/2.0/TCP 130.237.15.238:5060;branch=z9hG4bKnashds7
To: <sip:contextserver@130.237.15.238>
From: <sip:yusu@130.237.15.238>;tag=xfg9
Call-ID: 2010@130.237.238.168
CSeq: 8406 SUBSCRIBE
Max-Forwards: 70
Event: location
Accept: application/pidf+xml
Contact: <sip:yusu@kth.se>
Expires: 600
Content-Length: 0
```

The first line has two variables, the host name and the IP address of the context aware server. In this case, "contextserver" and "130.237.15.238" are the host name and the IP address of the server respectively. The second line indicates the transport means used for the transaction is UDP and identifies the location where the subscription is sent, i.e., the IP address of the server and a UDP port number. Here the server has the IP address 130.237.15.238 and UDP port number 5060. Header "To" and "From" are followed by the example of RFC 3856[43] that both two parties use the same SIP proxy, and the host name of the Pocket PC is extracted programmatically from the device. Call-ID is a field that needs to be random, so here we construct it using a random prefix concatenated with @ and the IP address of the watcher (i.e., the subscriber's IP address). Max-Forwards limit the number of hops a request can make on the way to its destination to be 70. The subscriber requests location events using "pidf+xml", indicating that an XML formatted PIDF file should be attached after the header of NOTIFY message. The header also specifies a contact and the expiration time of this subscription. As this SUBSCRIBE does not need any additional information beyond this, the SUBSCRIBE message ends with the line "Content-Length: 0".

There are 4 states for the application (including the states before and after the subscription). These are defined as state 1, state 2, state 3, and state 4. These correspond to the initial state, SUBSCRIBE state, listening for the 200 OK, and listening for a notification respectively. The state machine of the SIMPLE stack of the application is shown in figure 4.5. In this figure, the states in blue indicate inactive states while the states in gray are active ones.

Initially, the application starts in the initial state. This state is for the user configuration of the application, i.e., the user can set variables in the SUBSCRIBE message. Later the application goes into the SUBSCRIBE state and sends the SUBSCRIBE message to

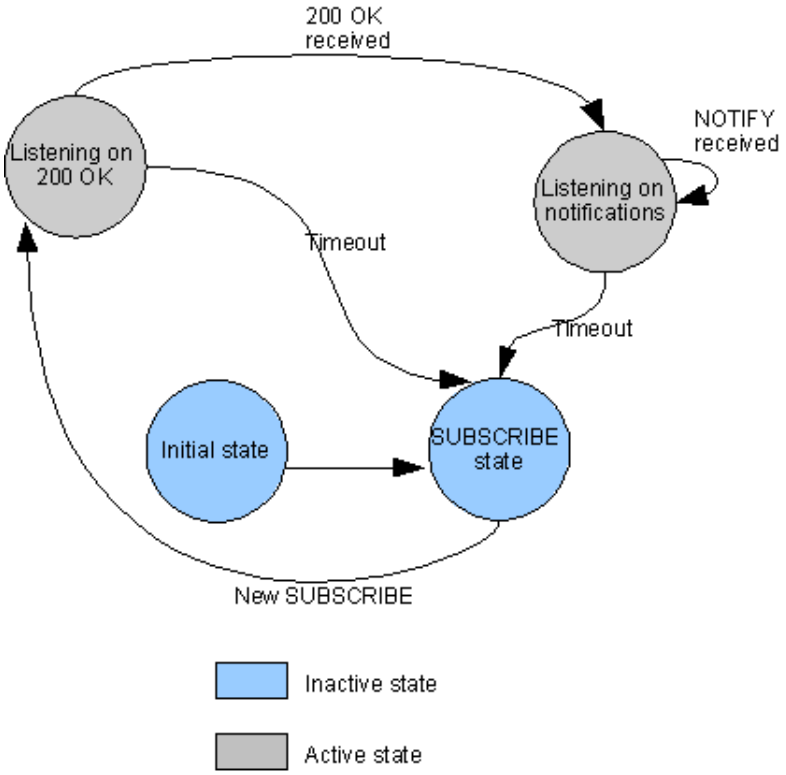


Figure 4.5: State machine of watcher application.

the SIP server which resides in the context aware infrastructure. After this subscription request, the application transitions to the third state: waiting for a 200 OK response - this response will be sent by the SIP server if the subscription was successful. Two events are able to change this state, the first one is because each SUBSCRIBE has an expiration time, when the subscription times out, the application will leave the current state and return to the SUBSCRIBE state. Or, if the 200 OK message is received, the application goes into the state of listening for notifications which is the fourth state. While listening for notifications, the application listens to the incoming NOTIFY messages and processes these NOTIFY messages until the SUBSCRIBE expires. When the SUBSCRIBE expires, the state machine returns into SUBSCRIBE state and sends a new subscription request.

After receiving a NOTIFY message, this NOTIFY message has to be parsed into two parts, the SIP header and the PIDF XML file. To extract the required values, the application has to parse the XML looking for the tag “location”. The actual code for doing this is shown as Appendix B.

4.5 Class design for context aware application

The idea of pervasive or ubiquitous computing differs from the traditional desktop paradigm in which a single user engages a single device for a specific purpose. Instead, many computational devices in the environment compute simultaneously for one or more users. Such a user, however, does not need to know which devices are actually computing, the only thing he or she is aware of is that they are "using" an integrated, abstracted service. When the user is "using" this service, the service could simultaneously utilize different devices to affect computing, while this service varies based upon the user and their context, thus forming a dynamic context aware (computing) environment.

The context aware infrastructure is responsible for retrieving and storing context data from different computational devices. In order to collect environmental data, sensors are used by these devices. These sensors include not only physical sensors, but also logical sensors which gather and fuse information from physical sensors or virtual sensors. Therefore, the context aware system is hierarchical, ranging from detection to abstraction of the context; this hierarchy is organized as several layers, as described in section 3.2.

Above the layer containing context repositories is the context aware server. In this thesis, at the application layer, context information also has its own hierarchy.

4.5.1 Collecting context information

Given our focus on a campus environment, the users consist primarily of students and staff. Here the context includes the user's profile and information about the buildings and rooms comprising the campus. Consider a room booking system, useful context includes the user's current location, the name of the building, the floors, room numbers or names, the number of people in a specific room, the room's booking status, etc.

When the user is using the room booking service (in this case based upon the open source project MRBS[56]), in order to provide more accurate information, the user's current location can be used to tailor the service. This means, the system should first learn the user's location in order to tailor its response based not only on the current time, but also upon the user's current location.

In this thesis, we have used the SIMPLE mechanism and created a dedicated class to request context information from the context aware infrastructure, each type of context information

can be retrieved separately and optionally. This class in the .Net framework is called "context_collector". By sending a SUBSCRIBE with different values in the "Event" option in the SIP header, the application can retrieve and store the updated context information from the context server.

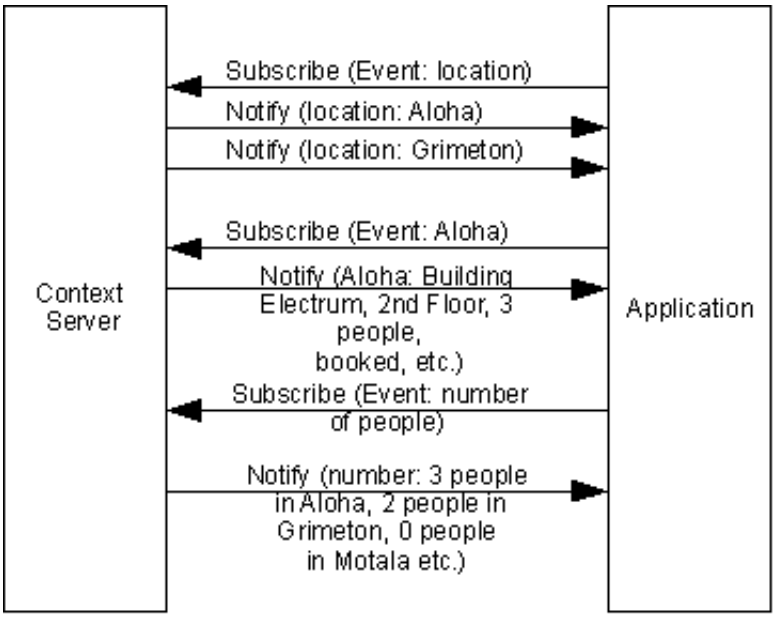


Figure 4.6: Example of retrieving additional context information from the context server.

Figure 4.6 shows an example where the user wants to book an available room which is closest to him. This example shows how the user's device requests context information from the context server. In this figure we omit the 200 OK and blank NOTIFY messages from the message exchange. First, the user's application subscribes to the server indicating he or she is interested in the device's current location, upon receiving this the server sends the current location of the user (Aloha) immediately. Since the user is walking through the building, as he approaches the room Grimeton, the sever discovers the change in the user's location and updates the application by sending a notification of this new location (Grimeton) to the user's application.

Whenever connectivity is available, the application can also ask for additional context information from the server. Here the device subscribes to the server putting "Aloha" in the event header. Indicating that the user wants to know additional information about Aloha. Consequently, the server looks in its database and sends the context related to this room as a single document. This information indicates that Aloha is on the second floor of Electrum, there are 3 people currently in the room, and the room is booked until 15:00, etc.

Since the user wants to book a vacant room, the device sends a new subscription with "occupancy" as the event of interest. This time, the server collects information from all the rooms and forms a new XML document, which shows that there 3 people in Aloha, 2 people in Grimeton, 0 people in Motala, then send this document to the user's application. This is different from sending a query looking for an *unbooked* room - since Motala might actually be booked, despite the fact that there are currently zero people in it.

Before collecting context information, it is essential that the user can learn which context information is available, this is necessary in order to ask for the **relevant** updated information. In this paper, we assume that the user has access to a virtual map of the building – which is displayed on the screen of his or her Pocket PC. The user can get such a map from the nearest web server, as was done for the ActiveCampus Explorer[2] which is described in section 2.2.2.

4.5.2 Context interpreter in the application

After getting the context information relevant to the user's vicinity, there are three ways for the context information to be used. One is to directly show the received information to the user, for example displaying the information about the room Aloha (Building Electrum, floor 2, 3 people are in the room, and the room is booked until 15:00). This requires the least computation and is very straightforward.

The second way for the context information to be used is to put this information directly into other functional classes. For example, in the location based reminder, the context "location" is used directly for the class "context_outlook" to manipulate the built-in outlook calendar of the Pocket PC.

The last way to use the new context information is to process it based upon an application's specific rules. Thus, instead of using the context information directly, a so called "context_interpreter" class gathers two or more elements of retrieved context information and combines them. Utilizing a function table, the "interpreter" fuses these elements of context information, creating a new element of context information, which is more abstract and can even be application specific context. For example, the class "meeting_probability" combines the location of the user (the room where the user is sitting) together with the number of people in the room, to compute the probability of a specific type of activity taking place e.g.,(meeting, group discussion, etc.). For example, if the number exceeds

3, there is a x% that the user is on a meeting. Therefore the output of the context interpreter is the context "In a meeting: x%". This context is highly valuable for other higher level applications, since this probability of meeting can in turn be an input to a further computation, for example, to enable the user's Call-Processing Language (CPL)[57] to determine if an incoming call should be diverted to voice mail or not. By monitoring network traffic a sensor might estimate (based upon the pattern of communication and the communicating parties) that someone is using a mobile device to remotely control a data projector which is being used to present a slide, because the "Projector in use" notification was generated. Combining these two elements ("In meeting: x%" and "Projector in use") may enable a SIP proxy to conclude that the user should not receive a phone call at this moment, hence the calls should be redirected to the user's voice mail or perhaps to another destination.

Bayes law could be used for the computation of the conditional probability:

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B|A)\Pr(A) + \Pr(B|\neg A)\Pr(\neg A)}$$

Let us say that the state *In a meeting* is A, *Projector in use* is B, the P(B|A) is the probability of *using a projector in a meeting* – this we can give some fixed value. Then the probability that *if the projector is in use that the user is in a meeting* (P(A|B)) can be calculated according to the equation. Note the probability that the user does not want to receive a call at the present can be deduced from given the present time, that the user is likely to actually be in a meeting (due to being in a meeting room, being scheduled for a meeting, and there are 3 or more people in the room), that the projector is in use, and that the user really does not want to receive calls while actually in a meeting. Hence, the P(A|B) can increase the accuracy of correctly handling an incoming call.

Figure 4.7 shows the class function designed for the Pocket PC, along with the interaction between the Pocket PC and the context aware server. As illustrated, the communication between the context server and the Pocket PC is based upon the SIMPLE protocol. The Pocket PC gets event updates by receiving NOTIFY messages. The context collector is realized by the class "context_collector", that is able to receive, parse, and extract the context value as a string. Above the context collector, is the low level context which be propagated in three different directions. On the left, "Display the context" is realized in an applications such as the room booking system where the context is shown to the user without being processed. The application (Application_1) in the middle is an application such

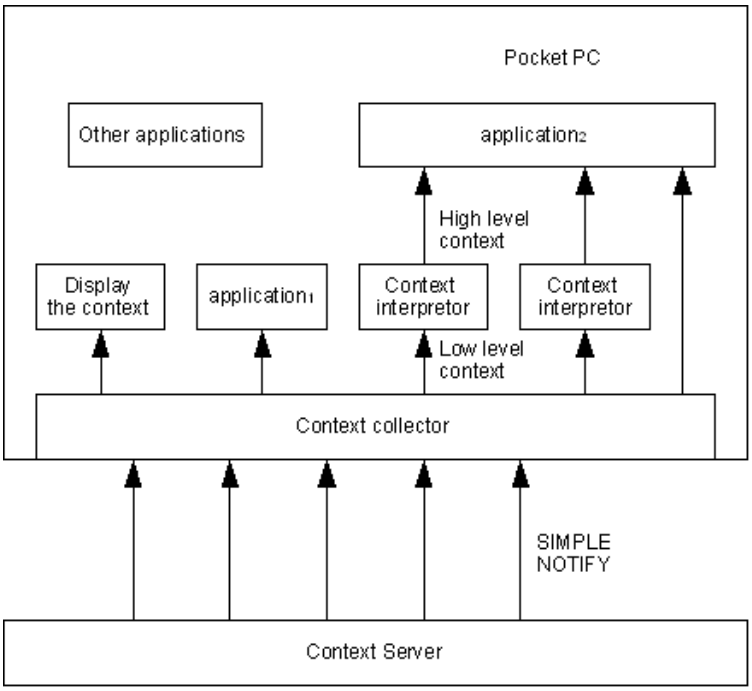


Figure 4.7: Architecture of several possible context aware applications.

as the class "context_outlook" that uses the low level context information directly (Here the context is invisible to the user - but what the user sees is affected by this context). On the right there are two context interpreters abstracting the low level context information into higher level context, later these different context elements are combined and used by another application (Application2).

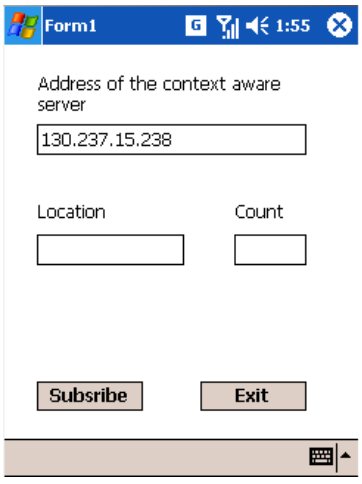


Figure 4.8: Graphical user interface for the location based reminder application.

Table 4.1 complements the Figure 4.7 by explicitly describing the classes that were implemented as part of this thesis project. Note that for the C# project implementation

on the Pocket PC, usually we create a main class in order to integrate the function classes and display results to the user. For the location based reminder, the class "location_based reminder" has a simple graphical user interface that enable the developer to keep track of the received messages. Also, it specifies how to call the function class "context_collector" and "context_outlook".

Figure 4.8 shows the simple user graphical design for the location based reminder. On the screen of the pocket PC, we inserted minimal components in order to keep the user interface simple. We have labels for each textbox, one textbox for the context aware server's IP address, while the other textbox displays the received location. Besides the location display, we imported a textbox for counting the number of location updates that the application has received, this can help the developer to evaluate and keep track of a device's location history.

Table 4.1: Class design of the thesis

Name of class	**Features and Functions**
context_collector	Responsible for collecting context by subscribing to the context aware server and retrieving the context by parsing NOTIFY messages received from the context aware server. Different types of context can be requested in SUBSCRIBE messages by utilizing different Event values.
context_outlook	An application-level class that uses the location context to manipulate the built in outlook calendar of the Pocket PC. The context "location" is transparent to the user, the user will only see the alarm from the built-in calendar.
room_booking	An application-level class that uses the context information (class location, user's location, number of people in the room, etc.) to display information to enable the user to choose the most suitable room and book it.
context_interpreter	A class that collects context information from the Context_retrieve class and generates higher layer context information.
location_based reminder	The main class which integrates context_outlook and context_collector that makes a complete application for receiving location update and notify the user if there is a corresponding appointment in that location.
room_booking_viewer	The main class which integrates room_booking and context_collector that makes a complete application for the user to view the room booking information and room occupancy.

Chapter 5

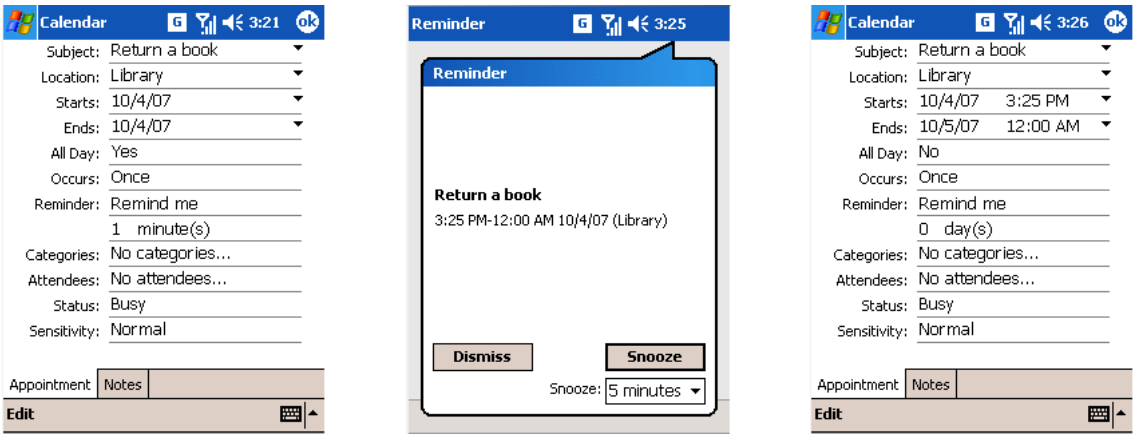
Evaluation

In this section we demonstrate and evaluate the performance of the proposed architecture for context aware applications. First we give an example of how the user is notified when he or she enters a specific location. Following this is an example of displaying the context information in conjunction with the room booking system. Finally we give an example which utilizes a context interpreter. The major focus in the evaluation part of this thesis is the evaluation of context information retrieved from the context aware infrastructure, more specifically, how much time it takes for each step, and what are the factors that affect this delay. The chapter concludes with a section that discusses some user satisfaction concerning the context aware applications which we have developed.

5.1 Location-based reminder

After the integration of the two classes (context_collector and context_outlook), the context aware location based reminder was tested and evaluated. The application first inputs "location" as the requested event to the context_collector, along with the IP address of the context aware server (we assume this application knows the IP address and the host name of the context-aware server - for this prototype this information is manually configured by the user, but it could be learned by a service location discovery protocol, such as Service Location Protocol[58]). Whenever a new NOTIFY message is received, if the value of the context information (i.e., the location) changes in the class of context_collector, then the class context_collector will call the class context_outlook once giving it the updated location information. Hence, the appointment entries will be modified based upon the current location of the device.

Figure 5.1 shows an example of how the user is notified when he passes by the library. Earlier in the day the user entered an appointment into their calendar (figure 5.1a), this appointment is to "return a book" in the location "library". The user set the "All Day" field, meaning that this appointment is valid for a whole day. The "Reminder" field is not relevant in this case since the starting time of this event has already passed (since the time is after 00:00 in the day).



(a) (b) (c)

Figure 5.1: Virtual scenario for the location-based reminder when the user passes by the library.

When the user passes by the library, the context infrastructure detects the RFID tag on the Pocket PC, the RFID reader tells the context-aware server that this user is at the library. When the server receives this, it generates a new NOTIFY message with the updated location information concerning the user and sends it to the user's device. At the device the application extracts the location information using the context_collector. Then by calling the class context_outlook, the device searches for an appointment in the Pocket PC's built-in calendar that is both valid and has the location "Library". If a match is found, then the context_outlook manipulates the entry to change the starting time to the current time, sets the reminder to remind the user 0 seconds before the event, while keeping the end time unchanged. Therefore, the user gets an alarm immediately when he approaches the library (figure 5.1b). Figure 5.1c shows the changes just before the user was notified, since the end time of the event has not changed, the calendar will interpret the current time to be when the user will still be reminded of this appointment.

5.1.1 Evaluation methodology

Generally speaking, the evaluation of a service should be divided into two parts. One is the performance of the system, while the other is to what extent the user is satisfied with using such a service and how much of a distraction or benefit it is.

As stated before, the major advantage of the location based reminder is that since the alarm comes from the built-in calendar of the Pocket PC, no matter what the user is doing with his device, the alarm will pop up and notify the user. Hence the user does not need to learn a new interface in order to be notified. Thus the factor that affects the user's satisfaction could be quantified in terms of the percentage of true positives, i.e., the user receives the correct notification at the correct location. When the location has changed and the user is notified of the change in location and it is one of the locations for which they have a reminder, then this outcome is regarded as true positive; when the location of the user has not changed and there is no reminder for this location, then the outcome is a true negative. But if the user receives a reminder when there has been no change in location or when the reminder is not appropriate for this location, then the outcome is regarded as either a false negative (which is the opposite of true positive) or a false positive. Also, when the user has changed the location but does not get the corresponding reminder, it is regarded as a false positive.

Table 5.1: Possible outcomes of the location based reminder

	The user gets the reminder (positive)	**The user gets a wrong reminder**	**The user does not get the reminder (negative)**
Location changes (true)	true positive	false positive	false negative
Location does not change (false)	false positive	Since there is no location change, "wrong" does not have a meaning.	true negative

In term of the system's performance, the criteria is split into the quality of service of each task of the system. When the server receives an updated event from a sensor, the relevant metric is the time delay from when the sensor's input changed. The time delay required for the device to generate an alarm after it receives the notification from the server, and percentage of true/false positives/negatives could be additional metrics. In this thesis, the metrics we are going to measure are the time delays from the location changing to the user's receiving notification (reminders) from their calendar, and the true/false positive/negative

ratios.

5.1.2 Location event generater

In order to evaluate the location based reminder application, changes in the device's location are needed. Additionally, it is necessary to keep track of when the location changes in order to compare it with the time when the server receives the event update and starts to process this event. This can be done by applications either on the server side or on the Pocket PC that records the time of these events, or by a third party who sniffs the communication channel (in order to collect and time stamp these events). In this thesis, we created an application that can generate (virtual) location update events and send them to the server at specific time intervals.

RFC 3903[59] introduced a PUBLISH mechanism (within the SIP framework) for event publication from a user agent to an entity which is responsible for processing and send the event to the interested parties. As a new SIP method, the PUBLISH mechanism for event update occurs between the PUA and PA. [43] describes this mechanism when the event concerns "presence". In addition to the "presence" event, this framework can be extended to support publication of any event state for which there exists an appropriate event package, as defined in [41]. Hence RFC 3903 introduces new terms for both entities that are participating in this communication. A User Agent Client (UAC) which publishes event state is termed an Event Publication Agent (EPA), note that the EPA acts as the PUA in [43]. While an Event State Compositor (ESC) processes the PUBLISH request, hence it is similar to the role of the PA for the "presence" event in [43]. The EPA represents sensors and sends an event update to the ESC as an event when there is a change in state. After receiving this event update and processing it, the ESC updates its database, and later sends a new NOTIFY message to the watcher who has subscribed to this event. Figure 5.2 shows an example of the message flow as is defined in [59].

After the watcher subscribes for an event notification, the ESC looks in its database for the event and sends a first notification. While the EPA can send the PUBLISH message whenever there is an available connection between itself and the ESC. Subsequently, the ESC sends the first NOTIFY to the watcher, then the EPA sends a PUBLISH message in order to update those entities who have indicated an interest in the state of this event. Note that in the PUBLISH header, there is a field indicating the expiration time of the PUBLISH. Both the ESC and the EPA should maintain a state in order to store this expiration time. When the event update expires, the EPA has to send a refresh PUBLISH in order for the

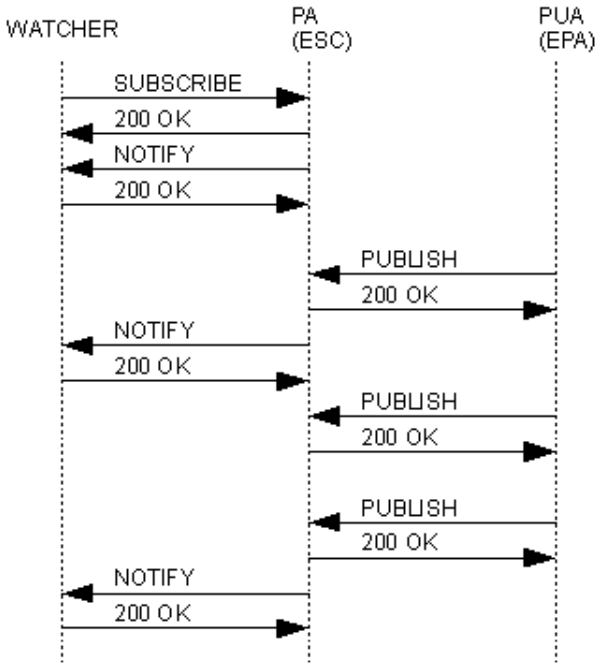


Figure 5.2: An example of the message flow for SIP PUBLISH.

ESC to learn the state of the event. In this figure, the second PUBLISH message is sent because the initial PUBLISH has expired. Since it is a refresh PUBLISH, the ESC does not process it and simply sends a new notification to the watcher. Later when there is an event change noted in the EPA, then the EPA sends a new PUBLISH to the ESC with the same event type and its updated value. This PUBLISH is regarded as the modify PUBLISH, whose mission is to modify the event data which was created by the previous PUBLISH message. Since the event has been updated, the ESC sends a new NOTIFY to the watcher, indicating that the event which the watcher is interested in has been updated.

When building the location event generator, we use the PUBLISH message (as the above example illustrated) to communicate between the virtual location event generator on the Pocket PC and the context aware server.

For each successful PUBLISH request, in the 2xx response there will be an entity-tag in the SIP-ETag header field which is generated and assigned by the ESC. For the updating PUBLISH messages after the initial PUBLISH request, each must include this entity-tag from the SIP-ETag field of the previous 2xx response and put it into a SIP-If-Match field. Also, in order to distinguish “Refresh”, “Modify”, and “Remove” publications, the content length field and the Expire values in the header will be different. Table 5.2 shows how these different PUBLISH requests are distinguished based upon the SIP-If-Match, the Body, and

the Expires values.

Table 5.2: Publication operations, adapted from [59]

Operation	**Body?**	**SIP-If-Match?**	**Expires Value**
Initial	yes	no	> 0
Refresh	no	yes	> 0
Modify	yes	yes	> 0
Remove	no	yes	$= 0$

For the initial and modify PUBLISH requests, the "body" is the PIDF file attached after the PUBLISH header. It is an XML file that includes the updated event information. For our prototype, within the <status> tag, we defined a new tag <location>, which contains more detailed information of the room, in order to convey the location event update to the context aware server. Since location is the only event that needs to be updated, an alternative is to include the location value in the <note> tag. Note since XML parsing is sensitive to the <> tags, we cannot simply put a new <location> tag in the <note>, instead, we use "$" to define the beginning and the end of the value (it is only one of the alternatives). Example PIDF files for both cases are shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<presence
xmlns="urn:ietf:params:xml:ns:pidf"
xmlns:location="http://it.kth.se/~yusu/schemas/yusu.xsd"
entity="sip:ccsleft@130.237.15.238">
<tuple id="qcjN6z">
 <status>
   <basic>open</basic>
   <location>
     <description>Open area lab in wireless@KTH</description>
     <room>lab</room>
     <floor>2</floor>
     <coordinates>
       <latitude>59_24'19.53''</latitude>
       <longtitude>17_56'59.61''</longtitude>
     </coordinates>
   </location>
 </status>
  <contact priority="0.8">sunyu</contact>
</tuple>
</presence>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<presence
xmlns="urn:ietf:params:xml:ns:pidf"
xmlns:location="http://it.kth.se/~yusu/schemas/yusu.xsd"
entity="sip:ccsleft@130.237.15.238">
<tuple id="qcjN6z">
 <status>
   <basic>open</basic>
 </status>
  <note>location: $lab$</note>
  <contact priority="0.8">sunyu</contact>
</tuple>
</presence>
```

To measure the performance of the context aware architecture in this thesis and the location based reminder running on a Pocket PC, in this generator, we define 4 different locations: "wireless@kth", "Lab", "library", and "cafeteria". Each of which appears as the current location in sequence and remains unchanged for different time intervals.

5.1.3 System performance

When the user enters a different location, the time between the actual change in location until the user is notified is the sum of the following elements:

- The time for the RFID reader (or other location detection mechanism) to detect the user's location and send the PUBLISH message to the server.
- The time interval that while server processes the PUBLISH message and sends the NOTIFY to the watcher (in these experiments, a Pocket PC).
- The time from when the user's device gets the NOTIFY from the server and until the user is given an alarm notification (the later is displayed on the Pocket PC).

For the first element, we use the location generator to simulate the scenario when the user enters a different location. This introduces a time delay due to the communication between the location event generator and the context server. For the second element, delay is determined by the performance of the SIP Express Router (which we have used as the context aware server), measurements of this performance can be found in M. Eslami's Master's thesis[53]. After the NOTIFY message is received, the Pocket PC processes the message and looks for a match of this new location with a "location" as specified for an

entry in the built-in calendar. Thus if the user enters a location where he or she has a corresponding unfinished event scheduled, then the user is notified by a notification displayed on the Pocket PC. The time granularity for retrieving the location context is set to 1 millisecond in the application program on the Pocket PC.

The alarm granularity for appointments on the HP iPAQ Pocket PC h5550 is one minute, which means that alarms are only generated in the beginning of the minute. Therefore the granularity of the reminder is also one minute, hence the user might not be notified of the alarm for up to one minute. With regard to the evaluation metrics, from the user's perspective, the true/false positive/negative ratio will vary in different locations. For example, if the user is passing by a library, the alarm might not actually be displayed to the user until after 60 seconds. If the Pocket PC reminds the user after 49 seconds, although the user gets the reminder when he or she has passed by the library, the result should be regarded as a false negative (the user does not get the reminder in the right location). On the other hand, if the user enters a cafeteria to have lunch, because the user will stay in the cafeteria for a long time, even if the Pocket PC reminds the user one minute later, it does not affect the user's satisfaction and the result should be a true positive. For the evaluation of the system performance, however, metrics should be consistent. Therefore, for the purposes of this evaluation we define the case where the location of the user changes and the user get the corresponding notification within less than one minute, the incident is regarded as a true positive. If the location changes and the user does not get the corresponding reminder, it is a false negative. Additionally, if the user did not change their location and gets the alarm within a minute, it is a false positive.

Assuming that the probability of receiving a NOTIFY message is the same for every second in a minute, then since the reminder only generates alerts at the beginning of a minute, then the mean time for the user to wait to get the actual notify should be 30 seconds. Thus the relationship between which second of the minute the NOTIFY is received and the waiting time before the user gets and alarm, should be shown in figure 5.3. In order to test this hypothesis, we performed a set of the measurements, as described below:

First, for every second of a minute, the location update should happen with equal probability. Hence we set the location updates to have an update interval of 61 seconds, then after sufficient time every second in a minute should experience roughly the same number of NOTIFY messages. We use Wireshark to record the network traffic in order to analyze the data packets. In this scenario we use three devices: two Pocket PCs, one for subscribing and generating notifications to the user, the other one for updating the location

event. The third device is a context aware server which processes the location updates and sends NOTIFY messages to the subscriber(s). The Network Time Protocol (NTP) is used to synchronize these devices to the same NTP server (ntp.kth.se). On the Pocket PC we installed a program called SP TimeSync v2.3[1], which is a free NTP client for Pocket PC 2003 and on the context aware server we simply set the Ubuntu Linux to synchronize its time with ntp.kth.se. After the time synchronization, we set appointments in the subscriber Pocket PC with the location "Lab" and the subject "TestLab", the location "Library" with the subject "Testlibrary", the location "Cafeteria" with the subject "Testcafeteria", and the location "Wireless@kth" with the subject "Testwireless". All of these appointments were set as "All day" appointments.

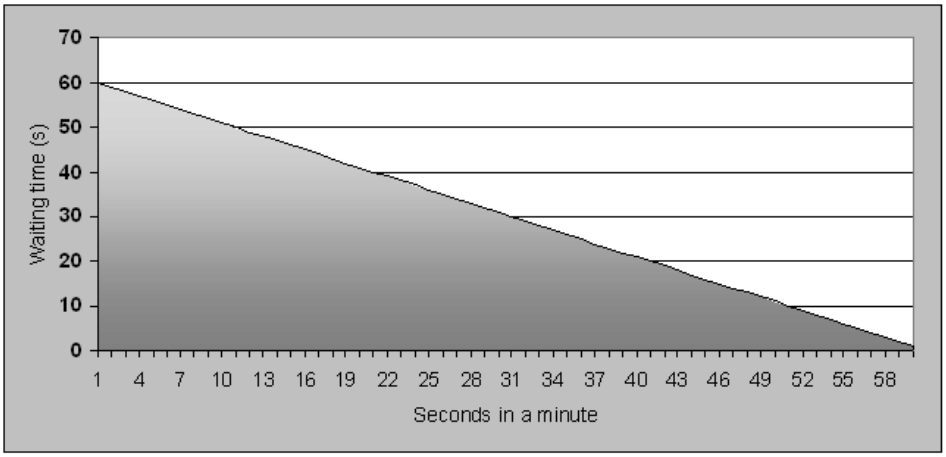


Figure 5.3: Assumed relationship between when the user changes theire location and the waiting time to get the notification.

Using the information contained in the Wireshark log file, we can plot the distribution of the waiting time before getting the notification for each second of the minute.

As the figure 5.4 shows, we can observe that the waiting time to get the alarm decreases as the second of the minute in which the user changes their location increases. But for the last 10 seconds in a minute (seconds 49 - 59), the user get the alarm almost immediately (within 2 seconds). Hence the mean time for a user to get the notification is 25.17s (here we use 1 second for delay in alarm during the last 10 seconds in a minute). For the cases when the user wants to get an immediate notification (less than 5 seconds delay), the fraction of true positives is 16.67%.

Within the Windows CE operating system, the notification subsystem uses the kernel alarm

[1]SP TimeSync v2.3, download available at http://www.freewareppc.com/communication/sptimesync.shtml, last visited 2007.11.17

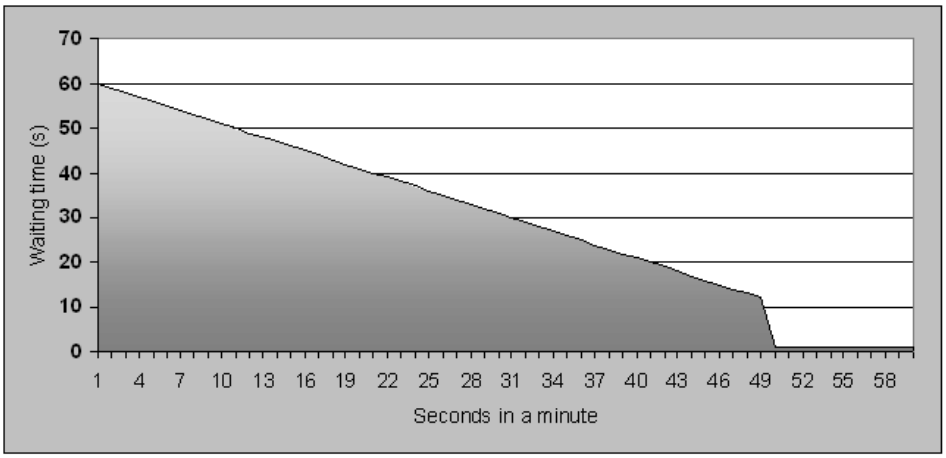


Figure 5.4: Measured relationship between when the user changes their location and the waiting time to get the notification.

to trigger time-based events. There is a variable called "dwNKAlarmResolutionMSec" in the OEM adaptation layer (OAL). This variable is set to 10 seconds as the default value, indicating that the resolution of the alarm notification is 10 seconds. Any event scheduled for a time sooner than that will be triggered immediately because the real-time clock sees the time (considering this resolution) to be the current time. While Windows CE allows this variable to be changed, in the range from 1 second to 60 seconds; OAL, which resides between the kernel of Windows CE operating system and the hardware of the device, is linked by the equipment vendor to the kernel libraries to create the kernel executable file. Therefore, once the device and the operating system has started, it is not possible to change this variable via the OAL since the initialization of the system has already occurred. It should be possible to change this value at run-time, but this would require kernel development - which is outside the scope of this project.

In conclusion, from the system perspective, the test result is satisfactory in that the sum of true positive ratio and true negative ratio is 100%, while the other two ratios (false positive, false negative) is 0.

5.2 Context aware room-booking

As mentioned in section 4.5.1, the context information indicating "number of people in the room" can be used by a room booking application in order to see that the room can be fully utilized. Usually a room booking system is a service that displays room information to a user along with the room's booking status; if the room is booked then it also conveys additional information including the starting time of the booking and the end time of the

booking. The original objective of such a service is to inform a person that the room is booked, thus it is not available during the booked period. Such room booking systems are widely used on university campus (as well as in other settings such as in a corporate setting).

Given the rising of the number of students in universities, meeting rooms are becoming a scare resource, especially during the examination period, when students often have a hard time finding rooms in order to have group discussions. Therefore, new methods of booking rooms should be introduced in order to enable the rooms to be used as efficiently as possible. Based upon personal observations, we should note that currently rooms are **not** used very efficiently. A common example of such situation is described below:

> *A group room in KTH Forum is booked from 14:00 - 17:00, but for some reason, the person who booked the room did not show up and use the room. However, since the room booking system still shows this room is booked, it is not available to others, hence this room will be wasted during the booked period.*

A new rule could be (similar to the rules of booking a laundry machine in most dorms or apartment buildings): If the person who booked the room does not show up within 15 minutes of the starting time of this booking, then the room booking is cleared and anyone can use this room. Using this new rule the allocation of rooms on a campus will be more efficiently utilized. This increases the efficient use of what otherwise would have been wasted resources, without requiring that a charging system for rooms be introduced (which is another way to encourage someone who books a room to actually use it or delete the booking).

Mapping the requirements on to a technical specification for a new room booking system, requires that in addition to the booking status and the room booking time, and duration – information about the (perhaps dynamically changing) number of people in the room is needed as well as the ability to know if the room is being used for the reason it was booked. Such a system could make use of the context aware infrastructure which is described in Daniel Hübinette's thesis[29] to provide the number of people in the room as context information. Note that this other thesis project is processing in parallel with this thesis project, so the room occupancy system is only a prototype and not yet installed at any of the meeting rooms. Therefore the examples in this thesis of room occupancy information are based upon manually entered occupancy values.

For every room booking system, we assume that there is a database residing on a server,

which stores the booking information associated with each room. This information includes Room ID, start time, end time, short description, id of this booking, and optionally a longer description of the booking. When the user wants to use this service, usually he or she can use a web browser (such as Microsoft's Internet Explorer, Mozilla's FireFox, etc.) to access the content which in the case of MRBS[56] is generated by a PHP script accessing a database (the database used was MySQL). Figure 5.5 shows an example of such a web page as seen via Microsoft's Internet Explorer for the MRBS room booking system as used at the Department of Communication Systems and the KTH Center for Wireless Systems (Wireless@KTH).

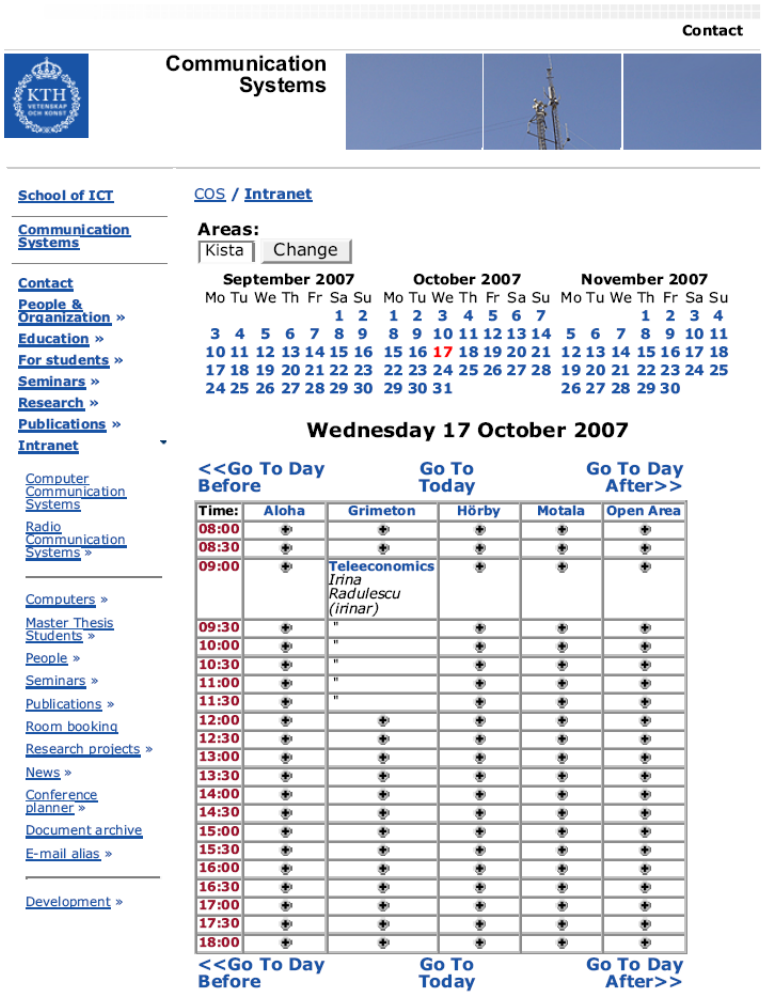


Figure 5.5: An example of the interface to a room booking system as viewed by a brower.

In this thesis, we use the C programming language to implement a program which directly accesses the MySQL database, to output an HTML document that only has the useful booking information concerning the rooms. Installing this application as a CGI script,

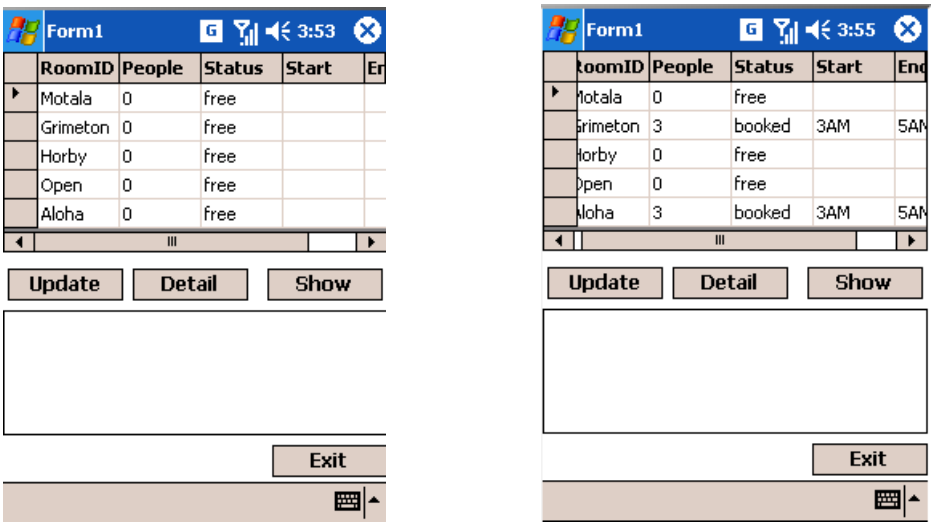
enables the scheduling information to be accessible via the HTTP server – without requiring that the user have special database access privileges. In our case, the new room booking application is implemented on the Pocket PC, although it could have been implemented elsewhere (for example at a proxy in the infrastructure). We use a C# program to establish a TCP connection between the Pocket PC and the web server in order to request that the CGI script be executed. After successfully established a TCP connection, we use an "HTTP GET" request to query the room booking systems for the schedule for the next 24 hours (thus providing us with information about current bookings). Finally we combine the booking information of each of the rooms along with the number of people in each room that we received from the context aware server, to build a table in the C# program. This table will subsequently be shown to the user.

The context "room_occupancy", i.e., the number of people currently believed to be in the room is included as a description in the <note> tag of the PIDF file; while the roomID is indicated in the <contact> tag. The resulting XML file as shown in the example below, shows that there are 3 people in the room "Grimeton".

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
xmlns:location="http://it.kth.se/~yusu/schemas/yusu.xsd"
entity="sip:ccsleft@130.237.15.238"> <tuple id="qcjN6z">
 <status>
   <basic>open</basic>
 </status>
  <note>3</note>
  <contact priority="0.8">Grimeton</contact>
</tuple>
</presence>
```

The user interface of the room booking viewer is shown in figure 5.6. As a demo application on the Pocket PC, it is limited to displaying the room occupancy for the rooms in Wireless@KTH. There are five rooms available in the database that can be booked, named "Motala", "Grimeton", "Horby", "Open area", and "Aloha". Each of these rooms has an associated row showing the room occupancy, the booking status, the start time of the book, and the end time of the booking. There are three buttons on the screen, when the user clicks "Show", the application initializes the table which contains all the rooms available and the initial value for each of the corresponding parameters, then it forms the display by inserting data into the component "Datagrid". As shown in figure 5.6a, all 5 rooms are displayed with empty values. Later, the user clicks the button "Update", causing the

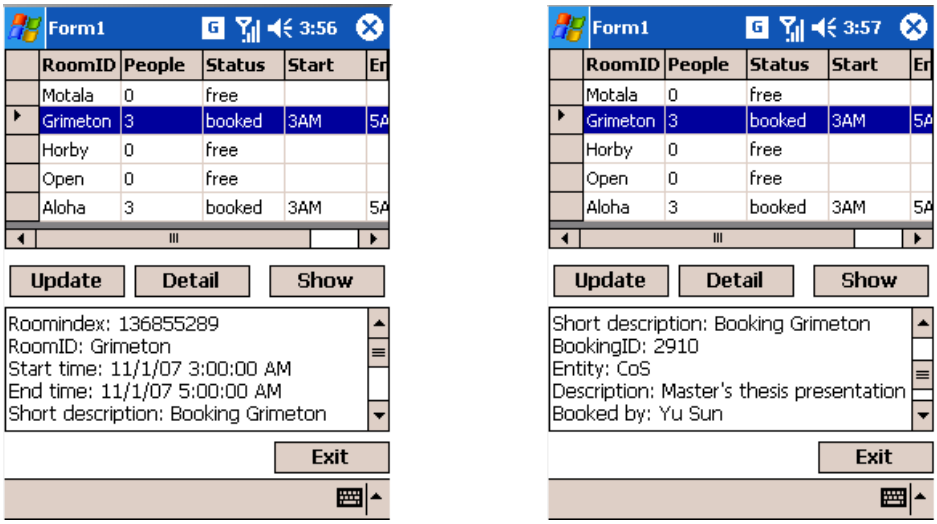
application to query the room booking database by sending an HTTP GET and receiving the corresponding reply. Additionally, the application also subscribes to the context aware server, requesting the event updates concerning the occupancy for each of these rooms. The server, receiving this request, should send a PIDF file containing the current information in the NOTIFY message. Hence, the application gets two XML files at this stage, one is the XML file from the query to the room booking system that contains the booking information for each room, while the other one is a PIDF file which contains the current occupancy information for each room. In order to simulate a room booking (for testing purposes) without actually accessing the actual room booking database in the department, here the HTTP GET returns a manually generated XML file, which includes two room bookings for the room "Grimeton" and "Aloha" respectively.



(a) (b)

Figure 5.6: A room booking viewer demo.

After the update, we can see the changes to the display (figure 5.6b), that the number of people in each room is displayed, note that based upon the supplied XML data for "Grimeton" and "Aloha", these rooms display their simulated booking information as well. Also, the "people" column of these two rooms show that there are 3 people in both rooms. The date grid only displays the basic information about the room booking, if the user wants to see the details, he or she could move the pointer to the column for the room, and click the button "Detail". As shown in figure 5.7, the details for a room booking is displayed. Since the Pocket PC has a touch screen, it is very easy to scroll down the text box which shows the detailed booking information. Thus the text message in figure 5.7b shows the continuation of the text in figure 5.7a.



(a) (b)

Figure 5.7: A room booking viewer demo (continue).

Given this room booking viewer, rooms can be used more efficiently. For example, if the room "Grimeton" is booked from 15:00 - 17:00 and the current time is later than 15:15, while there are 0 people in the room, then we can assume that this room is currently available.

It would be straight forward to extend this application from being a room booking viewer to an interface for actually booking rooms, but this is left as a future task. Note that one of the difficult issues here is to deal with authentication and authorization; i.e., is this user allowed to book this room at this time for this period.

It should be noted that while the department and the Wireless@KTH center both use the MRBS room booking system (note that there are separate MRBS room booking databases for several other departments, schools, and centers[2]), bookings of rooms outside of departments & centers within the greater university are booked generally through another system. Clearly a new booking application should be able to integrate these two systems, thus offering the user the ability to see (and perhaps) book any room in the university - for which they have the authorization to book. However, even without the ability to book rooms, the ability to located unused rooms across a multiple building campus would be very valuable and useful.

[2]MRBS is also used by by departments on the Valhallavägen Campus:
Biotechnology: http://intranet.biotech.kth.se/mrbs/day.php?day=01&month=12&year=2007
Computer Science and Communication: http://www.csc.kth.se/bokning/day.php?day=01&month=12&year=2007
Albanova (Physics, Biotechnology, and Astronomy): http://rooms.albanova.se/view_day.htm1?
EES also used MRBS, but as with CoS and Wireless@KTH it is only visible from within their intranet.

5.3 Context aware interpreter

A context aware interpreter translates low level context information into higher level context information, which can be sent to another application if needed. In this thesis, such a translation can be very straightforward. For example, after receiving the context information indicating the "room_occupancy", based upon a simple lookup a table, the interpreter calls a function which generates a new context "meeting probability". The table is quite simple as shown in table 5.3. However, as the context aware interpreter can be application specific and much more complex, also the data filled in this table is just an example, for the further work, the really relationship between the meeting probability and number of people in a meeting room should be well researched. One of the complex examples is in [60], where A. Devlic proposed an ontology model which extracts the context parameters of a person (locations, tasks, activities) and combines them in the OWL. Later this model is used by CPL script for decision makings.

Table 5.3: An example for context aware interpreter - the relationship between the number of people in a room and the probability for them to have a meeting

Number of people in a meeting room	**meeting probability (big room)**	**meeting probability (small room)**
0	0	0
1	0	0
2	10%	50%
3	25%	70%
4	30%	80%
5	40%	90%
6	50%	-
7	60%	-
8	70%	-
>8	80%	-

The context "meeting probability" is stored for the further use and is not immediately used after being generated. For higher layer applications such as CPL scripts, it is useful to combine this meeting probability along with the user's preferences, the user's tasks, activities etc. in order to make final decisions. However, the design and implementation of such an application lies outside the scope of this thesis project.

5.4 User's experience for the applications

For applications running Pocket PCs, the evaluation from the user's perspective is very important. First, we must consider the difficulty level of getting and installing such a service. Ideally the only thing that the user should need to do is download the executable file to his or her Pocket PC, and enable the wireless network connection so that the application can access the remote servers. Fortunately, the current application is very straight forward and does not have any dependencies; thus it can be installed with one click via a URL. Secondly, for location based reminders, as they are implemented as an extension to the current built-in calendar application, the user does not need to learn to use a new user interface. However, the room booking viewer application does require that the user learn a new user interface. The user may find it worthwhile learning to use this new simple interface, as it enables the user to query the booking information without requiring the use of a web browser, and the information can include the room occupancy, since that the current web based user interface to the room booking systems were not designed to be used on a device with a small screen for the reason that using them would either require lots of scrolling or require that a new interface be designed for devices with small screens.

Traditionally, pervasive computing engages many devices around the user to provide the user with information and events. The advantage of this architecture is that it releases the user being bound to one single device and significantly increases the available computation power and the accuracy of context-aware applications. However, there is always a tradeoff between processing data on a single device and on multiple devices, since sometimes the user might want to keep the raw data and use this data for a different purpose. Therefore, as long as it does not require a lot of processing power or time, from the end user's device (e.g., the mobile devices carried by the user) such devices could process the low level data in order to locally extract context information which could be used inside the device by applications (thus preserving the user's privacy and integrity). However, consideration of these issues lie outside the scope of this thesis.

Chapter 6

Conclusions and Future work

6.1 SIMPLE for event notification

This thesis successfully designed, implemented, and examined a context aware architecture based upon using SIMPLE. It introduced location as a new "event" for event updates, and introduced new XML tags for describing further details of such events in a PIDF file.

In order to receive the desired context information, the user (watcher) subscribes to the appropriate context aware server (presence agent), subscribing to the "event" that is relevant to the user. The context aware server, stores the user's context information and listens for corresponding event updates. Sensors (presence user agent) detects the changes to the user's context and generates updates to the context aware server. The server, acts as middleware between the user's application and these sensors.

6.2 Architecture within the user's device

Instead of requiring that the SIP Express Router perform all the computation, in this thesis, we proposed an architecture running in the application layer. As long as there is not much need for extensive processing power, the user can utilize layered context processing. Here, context in the user's mobile device is computed locally, and used by other services within the device. This enables context to be retrieved from the server with minimum delay, while also avoiding the server from the heavy load of service processing. The current solution of subscribing for all location context events associated with this user adds additional processing and communication burdens to the user's mobile device, but avoids revealing the details of the user's scheduled and what are the "significant" locations to the user.

There remain questions about distributing this context information beyond the device, but this is left as a topic for future work.

6.3 Location based reminder

A location based reminder application on the Pocket PC was designed and implemented. The basic idea was to extend the existing built-in calendar application on the Pocket PC in order to make it context (location) aware. When the user approaches a new location, after learning the user's location from the context aware server, the PDA checks in its calendar database to see if there is an appointment corresponding to this new location. If there is a match, the user will be notified of their scheduled "appointment" at the right location. Testing with simulated location updates indicates that the user always receives the correct alarm IF the user has connectivity to receive the NOTIFY messages from the context server. No measurements were main as to the behavior when the user is actually moving in and out of regions of WLAN connectivity. The only reliability in these notifications is that if the server does not receive an acknowledgement, it will continue to send NOTIFY messages following a backoff timer.

6.4 Room booking viewer

Based upon the same class design used earlier for retrieving context information, the room booking viewer application is able to get room occupancy updates while offering detailed room booking information. By combining room occupancy with room booking information (especially the starting time for the booking), the user is able to guess if there is a room available even when though a room has been booked, but is actually not in use. This could significantly improve the utilization of the rooms on a campus.

6.5 Limitations

6.5.1 User's perspective

For the earlier applications, there were limitations in the services. Several of these limitations could be addressed in future work.

First, for the location based reminder application, there is always a concern about power

management. Since the wireless connection is very power consuming, it is not a good strategy for the device to always be connected. One thing which can be done is to take advantage of the known latency in the actual alerts being presented to the user, hence the device can sleep until shortly before the next minute – but this requires timely delivery of the notifications when the device next comes online. Thus one of the elements of future work is to understand how to remain synchronized while allowing the WLAN link to be in sleeping mode when possible. This is a topic of an upcoming masters thesis project.

As shown in section 5.1.3, the Pocket PC has a resolution of 60 seconds for event notification delivery to the actual user via an alarm or calendar event notification. Therefore the mean time between a location update and the notification appearing on the screen is 25.17 seconds. Twice this value represents the granularity of the location based reminder application acceptable. However, this negatively affects the user's feeling of seamless as the user can pass by some locations (such as passing by a library even within a few seconds). As noted earlier one possibility is to decrease the timing resolution of alarms to decrease this latency.

Finally, for the room booking system viewer application, it is not yet clear how one could determine if a room is used by the person who booked it or if it is being used for the purpose for which it was booked. These remain as future work also.

6.5.2 Technical perspective

An application which implements a SIP client will use port 5060 as the default port number to communicate with the SIP Express Router, this leads to the problem that when there are several SIP applications running on the same device, as the packet received from this port may need to be delivered to one or more of these application. Alternatives are to utilize a proxy which keeps track of which incoming packet to this port should go to which application, a proxy which forward all incoming packets to all applications which are interested in incoming packets to this port number, or perhaps creating an integrated application which combines the functionality of all of these applications.

6.6 Future work

While this thesis offers a proof of concept application, the actual applications needs a lot of work in order to make it flawless. From the perspective of developers, this is a clear direction for future development which could build upon the current work. Along with

this is greater consideration of security, privacy, and integrity. Creating of a complete room booking client for the Pocket PC. Integration over the multiple room booking systems which are in use at KTH.

6.6.1 Kernel modification

In section 5.1.3, we saw the reason for the Pocket PC generating the notification immediately when it receives the NOTIFY message within the last 10 seconds of the minute. This suggest that a project is to change the value of the variable "dwNKAlarmResolutionMSec" (initialized in the OEM adaptation layer). The default value for this variable is set to 10 seconds, but it could be set between 1 second to 60 seconds. In order to access this variable, some kernel modification is needed, because it is not possible to easily change it once the operating system has already started.

When Windows CE starts, in the Boot Loader, it loads the image file (nk.bin), then invokes the function StartUp() which belongs to the OAL layer. Inside StarUp(), there is a an OEMInit() function which initiates all the hardware devices, the system tick, the real time clock, and the kernel independent transport layer. Therefore, to change the variable "dwNKAlarmResolutionMSec", developers should declare it in the OAL and set the variable to the desired resolution in the implementation of OEMInit() function. To improve the behavior of the location based reminder application, this value should be set to 60000 (60 seconds) milliseconds. Thus the true positive ratio will be significantly increased as the alarms will be more timely. Additionally, the implications of such a change should be understood. Alternatively another solution for increasing the timeliness of these alarms should be found.

6.6.2 Power management

The requirement for power management of a highly mobile device such as the Pocket PC differs from application to application. For the location based reminder, in order to save power, we need exploit the granularity of the event notification in allowing the device to wake up only when needed. While for the room booking viewer application, the tradeoff between sending the room occupancy information for all rooms in a single NOTIFY message or sending a separate NOTIFY for each room need to be further analyzed. Since in this paper we did not examine the power consumption or the processing required for handling different room occupancy NOTIFY messages. Power consumption for transmitting user bits via Bluetooth and WLAN from an HP iPAQ Pocket PC have been examined in a paper by

Alisa Devlic, et al. [61].

6.6.3 SIP proxy

The port conflict mentioned above in section 6.5.2 needs to be addressed. There are several approaches to solving this problem. The first solution is that we use different ports for each specific application. The problem with this is that since the SIP Express Router (acting as the context aware server) uses the same destination port number for all its modules, it is very difficult to change this destination port unless we rewrite (or reconfigure) each SIP module separately. The second solution is to introduce a SIP proxy in the user's mobile device, which locally translates the port number to distinguish between different SIP applications. Then (almost) no matter how many SIP application we have on a single device this proxy can make them appear to be unique, however, the cost is that all SIP messages have to be routed through this proxy. The second solution is better than the first since it is much simpler and does not require changes to the SIP Express Router. However, the detailed implications of such a local proxy are not yet understood, hence this remains as future work.

6.6.4 Security

As the applications are dealing with event notifications, the user specific data need to be protected. For example, although context information such as room occupancy and booking information might be publically available, the context related to a specific user such as this user's location, the user's behavior, who they are meeting with, etc. should generally be treated as private information. Therefore, when the user to preserve the privacy of context information, the communication between the user's device and the context aware server should be encrypted and further consideration needs to be given as to how to protect the privacy and integrity of this information.

When the device subscribes to the server, it indicates that it is interested in a specific context by specifying an "event" in the SIP header. The server listens to the corresponding updates and classify the updated event to each subscriber. The server should only send the context related to this subscriber, while avoiding spreading information about other users or incorrectly spreading this information to other users. See for example [62] and the current concerns about Facebook's Beacon advertising platform which passes information about a user - even when they are not actively logged into Facebook[1].

[1]Facebook, http://www.facebook.com/, last visited 2007.12.5

To prevent intrusion, both the communication between the server & subscriber, the server & the event updater should be protected. The easiest way to do this is implementing VPN tunnels between these three parties. If the event updater resides on a LAN, we can use layer 2 VPN such as Point-to-point tunneling protocol, Layer 2 Tunneling Protocol, or a layer three tunnel such as IPsec (IP security) between devices and the server to enhance security. For the channel between the mobile subscriber and the server, transport layer security (such as SSL or TLS) could also be used.

References

[1] Global Positioning System (GPS) (visited on July 1, 2007). [Online]. Available: http://www.gps.gov/

[2] W.G. Griswold, P. Shanahan, S.W. Brown, R. Boyer, M. Ratto, R.B. Shapiro, and T.M. Truong, "Activecampus: Experiments in community-oriented ubiquitous computing," *IEEE Computer*, vol. 37, pp. 73–80, October 2004.

[3] J. Kim and J. Huh, "Context-aware services platform supporting mobile agents for ubiquitous home network," *Advanced Communication Technology, 2006. The 8th International Conference*, vol. 1, pp. 136–139, 2006.

[4] T. Sohn, K.A. Li, G. Lee, I. Smith, J. Scott, and W.G. Griswold, "Place-Its: A Study of Location-Based Reminders on Mobile Phones (visited on July 19, 2007)." [Online]. Available: http://research.microsoft.com/~jws/pubfiles/ubicomp2005-placeits.pdf

[5] A.P. Patil, D.J. Kim, and L.M. Ni, "A study of frequency interference and indoor location sensing with 802.1lb and bluetooth technologies," *IEEE Wireless Telecommunications Symposium*, pp. 174–183, April 2006.

[6] J. Hightower and G. Borriello, "Location systems for ubiquitous computing," *IEEE Computer*, vol. 34, pp. 57–66, Auguest 2001.

[7] C. Schmandt and N. Marmasse, "User-centered location awareness," *IEEE Computer*, vol. 37, pp. 110–111, October 2004.

[8] A. K. Dey, "Understanding and using context," *IEE Personal and Ubiquitous Computing*, vol. 5, pp. 4–7, 2001.

[9] J. Pan, C. Tan, and W. Lee, "Context-aware service protocol. an extensible and configurable framework for user context awareness services in pervasive computing systems," *IEEE Wireless Communications and Networking*, vol. 3, pp. 2058–2063, 2003.

[10] J. Hightower and G. Borriello, "Location Sensing Techniques (visited on June 25, 2007)." [Online]. Available: http://portolano.cs.washington.edu/papers/UW-CSE-01-07-01.pdf

[11] S. Russell, "Nag yourself in a new way (visited on June 25, 2007)." [Online]. Available: http://www.swjournal.com/articles/2005/05/06/news/news22.txt

[12] N. Marmasse and C. Schmandt, "Location-aware information delivery with commotion," *IEE Handheld and Ubiquitous Computing. Second International Symposium*, pp. 157–171, 2000.

[13] Geominder (visited on June 26, 2007). [Online]. Available: http://ludimate.com/products/geominder/

[14] S. Kim, S. Park, J. Lee, Y. Jin, H. Park, A. Chung, S. Choi, and W. Choi. "Sensible appliances: applying context-awareness to appliance design," in: *IEE Personal and Ubiquitous Computing*, vol. 8, 2004, pp. 184–191.

[15] K.A. Li, T. Sohn, and W.G. Griswold, "Evaluating Location - Based Reminders (visited on June 25, 2007)." [Online]. Available: http://louis.ucsd.edu/~tsohn/pdf/reminders-techreport.pdf

[16] Geominder FAQ (visited on June 26, 2007). [Online]. Available: http://ludimate.com/support/geominder.php

[17] I. Smith, S. Consolvo, A. Lamarca, J. Hightower, J. Scott, T. Sohn, J. Hughes, G. Iachello, and G.D. Abowd, "Social disclosure of place: From location technology to communication practices," *IEE Pervasive Computing, third international conference*, pp. 134–151, 2005.

[18] P. Givy, "LBR-Location Based Reminder (visited on June 27, 2007)." [Online]. Available: http://home.snafu.de/phg/

[19] A. Schmidt, H. Gellersen, and C. Merz, "Enabling implicit human computer interaction: A wearable wid-tag reader," *IEEE Wearable Computers, The Fourth International Symposium*, pp. 193–194, October 2000.

[20] E.S. Bhasker, S.W. Brown, and S.W. Brown, "Employing user feedback for fast, accurate, low-maintenance geolocationing," *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications*, pp. 111–120, 2004.

[21] S. Pradham, C. Brignone, J. Cui, A. McReynolds, and M.T. Smith, "Websigns: Hyperlinking physical locations to the web," *IEEE Computer*, vol. 34, pp. 42–48, August 2001.

[22] P. Persson, F. Espinoza, and E. Cacciatore, "Geonotes: Social enhancement of physical space," *ACM Press Conference on Human Factors in Computing Systems*, pp. 43–44, 2001.

[23] Geonotes: Digital Graffiti in Public Places (visited on July 3, 2007). [Online]. Available: http://geonotes.sics.se/

[24] Wikipedia. Augmented reality (visited on July 3, 2007). [Online]. Available: http://en.wikipedia.org/wiki/Augmented_reality

[25] P. Persson, F. Espinoza, P. Fagerberg, A. Sandin, and R.Coster, "GeoNotes: A Location-based Information System for Public Spaces (visited on June 3, 2007)." [Online]. Available: http://www.sics.se/~petra/SocNavCh6.pdf

[26] W3C. Simple Object Access Protocol (visited on July 3, 2007). [Online]. Available: http://www.w3.org/TR/soap/

[27] R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The active badge location system," *IEE ACM Transactions on Information Systems*, vol. 10, pp. 91–102, 1992.

[28] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam, "Middleware for distributed context-aware systems," *IEE On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pp. 846–863, 2005.

[29] D. Hubinette, "Occupancy Sensor System for context-aware computing," Master's thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, December 2007.

[30] D. Salber, A.K. Dey, and G.D. Abowd, "The context toolkit: Aiding the development of context-enabled applications," *Proceedings of the CHI 99 Conference: CHI is the Limit - Human Factors in Computing Systems, ACM Press*, May.

[31] A.K. Dey, D. Salber, G.D. Abowd, and M. Futakawa, "An Architecture To Support Context-Aware Applications (visited on June 5, 2007)." [Online]. Available: ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-23.pdf

[32] A.K. Dey, G.D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *IEE Human-Computer Interaction*, vol. 16, pp. 97–166, 2001.

[33] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau "Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation 16 August 2006, edited in place 29 September 2006 (visited on July 3, 2007)." [Online]. Available: http://www.w3.org/TR/2006/REC-xml-20060816/

[34] H. Chen, T. Finin, and A. Joshi, "An Intelligent Broker for Context Aware Systems (visited on July 6, 2007)." [Online]. Available: http://www.cs.umbc.edu/~finin/ /papers/ubicomp03-poster.pdf

[35] M.K. Smith, C. Welty, and D.L. McGuinness, "OWL Web Ontology Language Guide (visited on July 6, 2007)." [Online]. Available: http://www.w3.org/TR/owl-guide/

[36] L. Kagal, T. Finin, and A. Joshi, "A policy language for a pervasive computing environment," *Proceedings of IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pp. 63–74, 2003.

[37] Adaptive and Context-Aware Services (ACAS): A project in the Affordable Wireless Services and Infrastructure program (visited on Auguest 10, 2007). [Online]. Available: http://www.wireless.kth.se/AWSI/ACAS/

[38] T. Strang and C. Linnhoff-Popien, "A context modeling survey," *First International Workshop on Advanced Context Modelling, Reasoning and Management (UBICOMP)*, September 2004.

[39] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261 (Proposed Standard), June 2002, updated by RFCs 3265, 3853, 4320. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[40] Cisco Documentation. Overview of SIP (visited on July 28, 2007). [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/product/software/ ios123/123cgcr/vvfax_c/callc_c/sip_c/sipc1_c/chapter0.htm

[41] A.B. Roach, "Session Initiation Protocol (SIP)-Specific Event Notification," RFC 3265 (Proposed Standard), June 2002. [Online]. Available: http://www.ietf.org/rfc/ rfc3265.txt

[42] J. Rosenberg, "SIMPLE made Simple: An Overview of the IETF Specifications for Instant Messaging and Presence using the Session Initiation Protocol (SIP)," Internet-Draft, 2007. [Online]. Available: http://www.ietf.org/internet-drafts/ draft-ietf-simple-simple-00.txt

[43] J. Rosenberg, "A Presence Event Package for the Session Initiation Protocol (SIP)," RFC 3856 (Proposed Standard), Aug. 2004. [Online]. Available: http: //www.ietf.org/rfc/rfc3856.txt

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

[45] M. Day, J. Rosenberg, and H. Sugano, "A Model for Presence and Instant Messaging," RFC 2778 (Informational), Feb. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2778.txt

[46] J. Rosenberg, "A Data Model for Presence," RFC 4479, 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4479.txt

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

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

[49] H. Khartabil, E. Leppanen, M. Lonnfors, and J. Costa-Requena, "Functional Description of Event Notification Filtering," RFC 4660, 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4660.txt

[50] H. Khartabil, E. Leppanen, M. Lonnfors, and J. Costa-Requena, "An Extensible Markup Language (XML)-Based Format for Event Notification Filtering," RFC 4661, 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4661.txt

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

[52] J. Urpalainen, "An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors," Internet-Draft, 2007. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-simple-xml-patch-ops-03.txt

[53] M.Z. Eslami, "A Presence server for Context-aware applications," Master's thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, December 2007.

[54] Microsoft Corporation. Pocket Outlook Object Model API (visited on June 6, 2007). [Online]. Available: http://msdn2.microsoft.com/en-us/library/aa454223.aspx

[55] S. Pratschner. Using POOM with .Net Compact Framework Whidbey Beta1 (visited on June 6, 2007). [Online]. Available: http://blogs.msdn.com/stevenpr/archive/2004/08/05/209390.aspx

[56] MRBS. (visited on November 10, 2007). [Online]. Available: http://mrbs.sourceforge.net/

[57] J. Lennox, X. Wu, and H. Schulzrinne, "Call Processing Language (CPL): A Language for User Control of Internet Telephony Services," RFC 3880, 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3880.txt

[58] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," RFC 2608, 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2608.txt

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

[60] A. Devlic, "Extending CPL with context ontology," *Mobile Human Computer Interaction (Mobile HCI 2006) Conference Workshop on Innovative Mobile Applications of Context (IMAC), Espoo/Helsinki, Finland*, September, 2006.

[61] A. Devlic, A. Graf, P. Barone, A. Mamelli, and A. Karapantelakis "Bluetooth and WLAN context distribution methods: An evaluation focused on battery power consumption," submitted to *the 9th International Conference on Mobile Data Management (MDM'08), Beijing, China*, April, 2008.

[62] A. Escudero-Pascual and G.Q. Maguire Jr., "Role(s) of a proxy in location based services," *13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications. (PIMRC 2002). Lisbon. Portugal*, September, 2002, Vol. 3, pp. 1252–1257.

Appendix A

Retrieving entries from Pocket PC's built-in outlook calendar

This appendix describes the steps to access the built-in outlook calendar and manipulate it. The Pocket PC device used in this case is the Hewlett-Packard Company (HP) iPAQ Pocket PC h5550 model[1] with the OS system Microsoft Pocket PC 2003. The sample code to access the calendar is written in Visual C# and runs under Microsoft corporation's .NET Compact Framework (version 2.0).

The hierarchy of a POOM object is illustrated below, it simply has four folders and several items.

- Application
  - Tasks Folder
    - Task 1
    - Task 2
    - Task 3
    - Task 4
  - Contacts Folder
    - Contact 1
    - Contact 2
    - Contact 3

[1]Gerald Q. Maguire Jr.'s notes on using the HP iPAQ h5550 (visited on October 2, 2007) http://web.it.kth.se/ maguire/ipaq-notes.html

        * Contact 4
    - Calendar Folder
        * Appointment 1
        * Appointment 2
        * Appointment 3
        * Appointment 4
    - Infrared Folder
        * Item 1
        * Item 2
        * Item 3
        * Item 4

Therefore, if we want to manipulate the items in Calendar, we only need to access the Calendar Folder.

For those who have Microsoft Visual Studio 8 installed, there are 3 steps to go through in older to create your own POOM application:

Step1: Download the POOM tlb file (pimstore.tlb) from the Windows Mobile blog[2]

Step 2: Reference the tlb file from your Visual Studio Project

- Right click on the "References" node for your project in the Solution Explorer and select "Add Reference...".
- Choose the "Browse tab" in the "Add Reference" dialog and select the pimstore.tlb file you downloaded in Step 1.
- In your references there is a "PocketOutlook" being added.

Step 3: Build the application

The code for retrieving the calendar items and display is shown below

[2] http://blogs.msdn.com/frankpr/archive/2005/06/22/431496.aspx

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using PocketOutlook;

namespace outlook1 {
    public partial class Form1 : Form
    {
        /***
         *Define the Applicationclass and Ifolder
         * ****/
        ApplicationClass outlookApp = null;
        IFolder calendarFolder = null;

        public Form1()
        {
            InitializeComponent();
            try
            {
                outlookApp = new ApplicationClass();
                ((IPOutlookApp)outlookApp).Logon(0);
                /***
                 * Bind calendarFolder to calendar
                 * ***/
                calendarFolder =
                outlookApp.GetDefaultFolder(OlDefaultFolders.olFolderCalendar);
                RefreshTaskList();
                ((IPOutlookApp)outlookApp).Logoff();
            }
            /***
             *Show exceptions
             * ***/
            catch (System.Runtime.InteropServices.COMException ce)
            {
                MessageBox.Show(String.Format
                ("Unable to Load POOM (0x{0})", ce.ErrorCode.ToString("8X")),
                    "Task Viewer");
                throw new ApplicationException();
            }
        }
        /***
         * Refresh the items display
         * ***/
        private void RefreshTaskList()
        {
            listBox1.Items.Clear();
```

```
for (int i = 0; i < calendarFolder.Items.Count; i++)
{
    IAppointment appointment =
    (IAppointment)calendarFolder.Items.Item(i + 1);
    String lvi = "";
    /***
     * Retrieve the location of the appointment
     * ***/
    lvi += appointment.Location;
    lvi += ", ";
    /***
     * Retrieve the subject of the appointment
     * ***/
    lvi += appointment.Subject;
    lvi += ", ";
    /***
     * Display the location and subject of the appointment
     * as one item
     * ***/
    listBox1.Items.Add(lvi);
    /***
     * Reset the string lvi
     * ***/
    lvi = "";
    /***
     * Retrieve the sensitivity of the appointment
     * and display
     * ***/
    switch (appointment.Sensitivity)
    {
        case OlSensitivity.olNormal:
            lvi += "Normal, ";
            break;
        case OlSensitivity.olPrivate:
            lvi += "Private, ";
            break;
        default:
            lvi += "  ";
            break;
    }
    /***
     * Retrieve the date of the appointment
     * and display
     * if reminder is set, display the date,
     * if reminder is not set,
     * display "No Remind"
     * ***/
    if (appointment.ReminderSet)
    {
        lvi += "Remind date:";
```

```
                    lvi += appointment.End.Date.ToShortDateString();
                    lvi += " ";
                }
                else
                {
                    lvi += "No Remind ";
                }
                listBox1.Items.Add(lvi);
            }
        }
        /***
         * Button to refresh the list
         * of appointments
         * ***/
        private void button2_Click(object sender, EventArgs e)
        {
            RefreshTaskList();
        }
        /***
         * Exit the application
         * ***/
        private void button1_Click(object sender, EventArgs e)
        {

            System.Windows.Forms.Application.Exit();
        }
    }
}
```

Appendix B

Source code of the class context_collector

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Net;
using System.Threading;
using System.Xml;

namespace location_based_reminder_final {

    public partial class Context_collector
    {
        private Thread th;
        public string context_type;
        private string msg;
        private UdpClient receiveUdp;
        private IPEndPoint ip_send;
        private IPEndPoint ip_listen;
        private Socket aserver;
        private IPAddress test;
        private byte[] recData;
        private byte[] data;
        private string SipHeader = "";
        private string toAddress = "";
        private string myAddress = "";
        private string branch = "";
        private string fromTag = "";
```

```
private string callID = "";
private string cSeq = "";
private string targetIP = "";
private int ifsip;
private int RandomNum_cSeq;
private int state = 0;
/***  state pointer for state machine
 *    0: initial
 *    1: subscribe sent
 *    2: ok recieved
 *    3: notification recieved
 *    4: ok sent
 *    5: subscription expires
 *    6: ok or notification with wrong CallID received
 *  ***/
private string messageType = "";
private string[] sb;
private string HeadLine;
private string viaLine;
private string FromLine;
private string CallIDLine;
private string SubscriptionStateLine;
private string CSeqLine;
private string SubLineMsg1;
private string subscripstate;
private string xmldata;
private string xmlvalue;
private XmlReader ContextReaderAgent;
private int xmlstartpoint;

public Context_collector(string context, string contextserveraddr)
{
    xmlvalue = "";
    /*
     * Get the type of context from outside classes
     */
    context_type = context;
    /*
     * Start the thread of listening for incoming
     * UDP packets
     */
    th = new Thread(new ThreadStart(Listen));
    th.Start();
    /*
     * Generate the random cSeq,
     * callID for SIP SUBSCRIBE hearder
     */
    Random Rnd = new Random();
    /*
     * Initialize cSeq, according to RFC 3261,
```

```
 * for non-REGISTER requests,the value can be
 * arbitrary as long as it is less than 2**31
 * Here we use 99999 as the maximum number for
 * intitialization
 */
RandomNum_cSeq = Rnd.Next(1, 99999);
cSeq = RandomNum_cSeq.ToString();
/*
 * Initialize the random part for the callID,
 * since the in c#, the "random" function generates
 * numbers based upon current time, this number
 * is also random.
 *
 */
callID = CreateRandom(11);
/*
 * the callID in SIP requst should be
 * unique, hence here we add the current
 * time stamp after the random number
 */
callID += System.DateTime.Now.ToString();
/*
 * Get the IP address of the context aware
 * server and use it as the destination IP address
 * for SUBSCRIBE
 */
test = IPAddress.Parse(contextserveraddr);
/*
 * Send the UDP packet to port 5060 which is the port
 * for SER in the context aware server
 */
ip_send = new IPEndPoint(test, 5060);
aserver =
new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
data = new byte[1024];
/*
 * Config the variables in the SUBSCRIBE header
 */
toAddress = contextserveraddr;
String strHostName = System.Net.Dns.GetHostName();
IPHostEntry ipEntry = System.Net.Dns.GetHostEntry(strHostName);
IPAddress[] addr = ipEntry.AddressList;
myAddress = addr[addr.Length - 1].ToString();
/*
 * Send the first SUBSCRIBE
 */
/*
     * Generate the random string for fromTag and branch
     * apart from the compulsory part"z9hG4bK"
```

```
            * The branch must be unique across space and time for
            * all requests sent by the UA
            * Here the numbers are examples
            */
          branch = CreateRandom(14);
          fromTag = CreateRandom(8);
          /*
           * Generate SUBSCRIBE message
           */
          SipHeader =
    "SUBSCRIBE " + "sip:ccsleft@" + toAddress + " SIP/2.0" + "\r\n" +
    "Via: " + "SIP/2.0/UDP " + toAddress +
    ":5060;branch=z9hG4bK" + branch + "\r\n" +
    "To: <sip:ccsleft@" + toAddress + ">" + "\r\n" +
    "From: <sip:" + strHostName + "@" + toAddress + ">;tag=" + fromTag + "\r\n" +
    "Call-ID: " + callID + "@" + myAddress + "\r\n" +
    "CSeq: " + cSeq + " SUBSCRIBE" + "\r\n" +
    "Max-Forwards: 70" + "\r\n" +
    "Event: " + context_type + "\r\n" +
    "Accept: application/pidf+xml" + "\r\n" +
    "Contact: <sip:" + strHostName + "@" + myAddress + ">" + "\r\n" +
    "Expires: 600" + "\r\n" +
    "Content-Length: 0" + "\r\n\r\n";
          /*
           * Send SUBSCRIBE to the context aware server
           */
          data = Encoding.ASCII.GetBytes(SipHeader);
          aserver.SendTo(data, data.Length, SocketFlags.None, ip_send);
          state = 1;
      }
          /*
           * for other classes to get the context
           * (when context_type = location, it reads the information
           * inside <location>"
           */
      public string get_context()
      {
          return xmlvalue;

      }
          /*
           * Thread for listening incoming UDP packets
           */
      private void Listen()
      {
          while (true)
          {
              try
              {
                  ip_listen = new IPEndPoint(IPAddress.Any, 5060);
```

```
    receiveUdp = new UdpClient(5060);
    recData = receiveUdp.Receive(ref ip_listen);
    receiveUdp.Close();
}
catch (Exception ex)
{ }
msg = Encoding.ASCII.GetString(recData, 0, recData.Length);
/*
 * Actions when there is a incoming UDP package
 */
if ((msg != null) && (state != 0))
{
    /*
     * See if it is a SIP message
     */
    ifsip = msg.IndexOf("SIP/2.0");
    if (ifsip <= 0)
    { }
    else
    {
        /*
         * Split the message
         */
        int i = msg.IndexOf(" ");
        messageType = msg.Substring(0, i);
        sb = msg.Split(new char[] { '\n' });
        i = 12;
        for (int j = 0; j < i; j++)
        {
            sb[j] = sb[j].Substring(0, sb[j].Length - 1);
        }

        HeadLine = sb[0];
        viaLine = sb[1];
        CallIDLine = sb[5];

        string SubLineCallID = CallIDLine.Substring(9);
        int callidIndex = SubLineCallID.IndexOf('@');
        string receivedcallID =
        SubLineCallID.Substring(0, callidIndex);

        if (receivedcallID == callID)
        {
            /*
             * Receives a 200 OK message
             */
            if (messageType == "SIP/2.0")
            {
                state = 2;
                /*
```

```
                            * According to SIP RFCs, there are some fields in
                            * the SIP message should be checked in order for
                            * the Subscriber to know it is valid
                            */
                        }
                        /*
                         * Receives a NOTIFY message
                         */
                        if (messageType == "NOTIFY")
                        {
                        /*
                         * According to SIP RFCs, there are some fields in
                         * the SIP message should be checked in order for
                         * the Subscriber to know it is valid
                         */
                        state = 3;
                        FromLine = sb[3];
                        i = FromLine.IndexOf('@');
                        SubLineMsg1 = FromLine.Substring(i + 1);
                        i = SubLineMsg1.IndexOf('>');
                        targetIP = SubLineMsg1.Substring(0, i);
                        CSeqLine = sb[4];
                        SubscriptionStateLine = sb[11];
                        subscripstate =
                        SubscriptionStateLine.Substring(20,
                        SubscriptionStateLine.Length - 20);
                            if (subscripstate == "terminated;reason=timeout")
                            {
                                state = 5;
                            }
                        }
                    }
                    else {
                        state = 6;
                    }
                }
                this.SendOK_SetText(msg);
            }
        }
    }

    private void SendOK_SetText(string text)
    {
           /*
            * Parse the PIDF file attached to NOTIFY message
            */
        if (state == 3)
        {
            xmlstartpoint = text.IndexOf("\r\n\r\n");
            xmlstartpoint += 4;
```

```
                xmldata = text.Substring(xmlstartpoint);
                xmlvalue = getcontext(xmldata);
                /*
                 * Send 200 OK back
                 */
                sendOKmessage();
                /*
                 * Clean the msg after recieving NOTIFY
                 */
                msg = "";
            }
            if (state == 5)
            {
                sendOKmessage();
                sendSubscribe();
            }
            else
            {   /*
                 * Clean the msg after recieving 200 OK
                 */
                msg = "";
            }
        }

        private void sendOKmessage()
        {
            /*
             * Send 200 OK back
             */
            String strHostName = System.Net.Dns.GetHostName();
            IPHostEntry ipEntry = System.Net.Dns.GetHostEntry(strHostName);
            IPAddress[] addr = ipEntry.AddressList;
            myAddress = addr[addr.Length - 1].ToString();
            String SIPok;

                SIPok =
"SIP/2.0 200 OK" + "\r\n" +
sb[1] + ";received=" + myAddress + "\r\n" +
sb[2] + "\r\n" +
sb[3] + "\r\n" +
sb[4] + "\r\n" +
sb[5] + "\r\n" +
"Content-Length: 0" + "\r\n\r\n";


            IPAddress test = IPAddress.Parse(targetIP);
            ip_send = new IPEndPoint(test, 5060);
            data = Encoding.ASCII.GetBytes(SIPok);
            aserver.SendTo(data, data.Length, SocketFlags.None, ip_send);
            state = 4;
```

```
}

/*
 * Resend SUBSCRIBE when it expires
 */
private void sendSubscribe()
{
    Random Rnd = new Random();
    /*
     * Initialize cSeq, according to RFC 3261,
     * for non-REGISTER requests,the value can be
     * arbitrary as long as it is less than 2**31
     * Here we use 99999 as the maximum number for
     * intitialization
     */
    RandomNum_cSeq = Rnd.Next(1, 99999);
    cSeq = RandomNum_cSeq.ToString();
    /*
     * Initialize the random part for the callID,
     * since the in c#, the "random" function generates
     * numbers based upon current time, this number
     * is also random.
     *
     */
    callID = CreateRandom(11);
    /*
     * the callID in SIP requst should be
     * unique, hence here we add the current
     * time stamp after the random number
     */
    callID += System.DateTime.Now.ToString();
    /*
     * Send the UDP packet to port 5060 which is the port
     * for SER in the context aware server
     */
    ip_send = new IPEndPoint(test, 5060);
    aserver =
    new Socket(AddressFamily.InterNetwork,
    SocketType.Dgram, ProtocolType.Udp);
    data = new byte[1024];
    /*
     * Config the variables in the SUBSCRIBE header
     */
    String strHostName = System.Net.Dns.GetHostName();
    IPHostEntry ipEntry = System.Net.Dns.GetHostEntry(strHostName);
    IPAddress[] addr = ipEntry.AddressList;
    myAddress = addr[addr.Length - 1].ToString();
    /*
     * Send the first SUBSCRIBE
     */
```

```
        /*
             * Generate the random string for fromTag and branch
             * apart from the compulsory part"z9hG4bK"
             * The branch must be unique across space and time for
             * all requests sent by the UA
             * Here the numbers are examples
             */
        branch = CreateRandom(14);
        fromTag = CreateRandom(8);
        /*
         * Generate SUBSCRIBE message
         */
        SipHeader =
"SUBSCRIBE " + "sip:ccsleft@" + toAddress + " SIP/2.0" + "\r\n" +
"Via: " + "SIP/2.0/UDP " + toAddress +
":5060;branch=z9hG4bK" + branch + "\r\n" +
"To: <sip:ccsleft@" + toAddress + ">" + "\r\n" +
"From: <sip:" + strHostName + "@" + toAddress + ">;tag=" + fromTag + "\r\n" +
"Call-ID: " + callID + "@" + myAddress + "\r\n" +
"CSeq: " + cSeq + " SUBSCRIBE" + "\r\n" +
"Max-Forwards: 70" + "\r\n" +
"Event: " + context_type + "\r\n" +
"Accept: application/pidf+xml" + "\r\n" +
"Contact: <sip:" + strHostName + "@" + myAddress + ">" + "\r\n" +
"Expires: 600" + "\r\n" +
"Content-Length: 0" + "\r\n\r\n";
        /*
         * Send SUBSCRIBE to the context aware server
         */
        data = Encoding.ASCII.GetBytes(SipHeader);
        aserver.SendTo(data, data.Length, SocketFlags.None, ip_send);
        state = 1;
    }
    /*
     * Function for generating random strings
     */
    public static string CreateRandom(int Length)
    {
        string _allowedChars =
        "abcdefghijkmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
        Random randNum = new Random();
        char[] chars = new char[Length];
        int allowedCharCount = _allowedChars.Length;

        for (int i = 0; i < Length; i++)
        {
            chars[i] =
            _allowedChars[(int)((_allowedChars.Length) * randNum.NextDouble())];
        }
```

```
            return new string(chars);
        }
        /*
        * Function for parsing context value in XML tags
        */
        private string getcontext(string dataFeed)
        {
            ContextReaderAgent =
            new XmlTextReader(new System.IO.StringReader(dataFeed));
            string context = "";
            if (context_type == "location")
            {
                while (ContextReaderAgent.Read())
                {
                    if (ContextReaderAgent.NodeType == XmlNodeType.Element)
                    {
                        if (ContextReaderAgent.LocalName.Equals("description"))
                        {
                            context = ContextReaderAgent.ReadString();
                        }
                    }
                }
            }
            else
            {
                while (ContextReaderAgent.Read())
                {
                    if (ContextReaderAgent.NodeType == XmlNodeType.Element)
                    {
                        if (ContextReaderAgent.LocalName.Equals(context_type))
                        {
                            context = ContextReaderAgent.ReadString();
                        }
                    }
                }
            }
            return context;
        }
        /*
        * Function for closing the class
        */
        public void stop_context_collector()
        {
            this.receiveUdp.Close();
            this.th.Abort();
            this.aserver.Close();
        }
    }
}
```

Appendix C

Source code of the class context_outlook

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using PocketOutlook;

namespace Location_outlook {

    public class Context_outlook
    {
        /***
         * Create instance for ApplicationClass
         * and IFolder
         ***/
        ApplicationClass outlookApp = null;
        IFolder calendarFolder = null;

        public Context_outlook(string context)
        {
            try
            {
                SetText(context);
            }

            catch (System.Runtime.InteropServices.COMException ce)
            {
                MessageBox.Show(String.Format("Unable
                to Load POOM (0x{0})", ce.ErrorCode.ToString("8X")),
                    "Task Viewer");
                throw new ApplicationException();
            }
```

```
}
/***
 * Main function
 * Only manipulate the calendar
 * This mechanism can be used
 * to manipulate the tasks,contacts,
 * and infrared items also
***/
private void SetText(string location)
{
    /***
     * Login to outlook
    ***/
    outlookApp = new ApplicationClass();
    ((IPOutlookApp)outlookApp).Logon(0);
    calendarFolder =
    outlookApp.GetDefaultFolder(OlDefaultFolders.olFolderCalendar);
    /***
    * Search for Items in the calendar folder
   ***/
    for (int i = 0; i < calendarFolder.Items.Count; i++)
    {
        IAppointment appointment =
        (IAppointment)calendarFolder.Items.Item(i + 1);
        /***
         * Judge if there is an application has the location field
         * which matches the incoming context
         ***/
        if ((appointment.Location == location)
        && (appointment.Sensitivity == OlSensitivity.olNormal))
        {
            /***
             * Judge if there is an application
             * valid in the current time
             ***/
            if (((appointment.Start <= System.DateTime.Now)
                && (appointment.End >= System.DateTime.Now))
             /***
              * Judge if there is an application valid
              * in the current time
              * and also is an AlldayEvent
             ***/
            || (((DateTime.Compare(appointment.Start,
            System.DateTime.Today) <= 0)
            && (DateTime.Compare(appointment.End,
            System.DateTime.Today) >= 0))
            && (appointment.AllDayEvent)
            ))
            {
             /***
```

```
                    * Manipulate the end time of appointment if
                    * it is a AlldayEvent
                   ***/
                      if (appointment.AllDayEvent)
                      {
                          appointment.AllDayEvent = false;
                          appointment.End = appointment.End.AddHours(23);
                          appointment.End = appointment.End.AddMinutes(59);
                      }
                      /***
                       * Keep the end time of the appointment unchanged
                       ***/
                      else
                      {
                          appointment.End = appointment.End;
                      }
                      /***
                       * Change the starting time of the appointment to the
                       * current time, thus the user can get the alarm
                       * immediately
                       ***/
                      appointment.Start = System.DateTime.Now.AddMinutes(1);
                      appointment.ReminderMinutesBeforeStart = 0;
                      /***
                       * Save the changes
                       ***/
                      try
                      {
                          appointment.Save();
                      }
                      catch (System.Exception ex)
                      {
                      }
                  }
              }
          }
          /***
           * Logoff outlook
           ***/
          ((IPOutlookApp)outlookApp).Logoff();
      }
  }
}
```

Appendix D

Source code of the location updater

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Net;
using System.Threading;

namespace location_updater {
    public partial class Form1 : Form
    {
        delegate void SetTextCallback(string text);
        private Thread th;
        private string msg;
        private UdpClient receiveUdp;
        private byte[] recData;
        private IPEndPoint serverip;
        private IPEndPoint ip_listen;
        private Socket aserver;
        private string cSeq;
        private string callID;
        private string[] location;
        private string sipifmatch;
        private string sipetag;
        private int state;
        /***
         * 0: Initial, no publish sent
         * 1: One publish sent
         * 2: One publish expires
         * 3: new event come, send new publish
         ***/
```

```
private int okstate = 0;
/***
 * 0: 200 OK not received
 * 1: 200 OK received
 ***/
private String messageType = "";
private int ifsip;
private System.Windows.Forms.Timer myTimer;
private int locationint = 0;

public Form1()
{
    InitializeComponent();
    myTimer = new System.Windows.Forms.Timer();
    myTimer.Interval = 1000;
    myTimer.Enabled = true;
    th = new Thread(new ThreadStart(Listen));
    th.Start();
    IPAddress test = IPAddress.Parse(textBox1.Text);
    serverip = new IPEndPoint(test, 5060);
    location = new string[4];
    location[0] = "Lab";
    location[1] = "Library";
    location[2] = "Cafeteria";
    location[3] = "Wireless@kth";
}

private void Listen()
{
    while (true)
    {
        try
        {
            ip_listen = new IPEndPoint(IPAddress.Any, 5060);
            receiveUdp = new UdpClient(5060);
            recData = receiveUdp.Receive(ref ip_listen);
            receiveUdp.Close();
        }
        catch (Exception ex)
        { }
        msg = Encoding.ASCII.GetString(recData, 0, recData.Length);
        if ((msg != null) && (state == 1))
        {

            ifsip = msg.IndexOf("SIP/2.0");
            if (ifsip < 0)
            { }
            else
            {
                int i = msg.IndexOf(" ");
```

```
                    messageType = msg.Substring(0, i);

                    if (messageType == "SIP/2.0")
                    {
                        string[] sb;
                        sb = msg.Split(new char[] { '\n' });
                        /*
                        i = sb.Length;
                        for (int j = 0; j < i; j++)
                        {
                            sb[j] = sb[j].Substring(0, sb[j].Length - 1);
                        }
                        */
                        string CallIDLine = sb[4];

                        string SubLineCallID = CallIDLine.Substring(9);
                        int callidIndex = SubLineCallID.IndexOf('@');
                        string receivedcallID =
                        SubLineCallID.Substring(0, callidIndex);
                        if (callID == receivedcallID)
                        {
                            okstate = 1;
                            string sipetagline = sb[7];
                            i = sipetagline.Length;
                            i = i - 11;
                            sipetag = sipetagline.Substring(10, i);
                            sipifmatch = sipetag;
                        }

                    }

                }
                this.SendOK_SetText(msg);
            }
        }
    }

    private void SendOK_SetText(string text)
    {
        // InvokeRequired required compares the thread ID of the
        // calling thread to the thread ID of the creating thread.
        // If these threads are different, it returns true.
        if (this.textBox1.InvokeRequired || this.textBox2.InvokeRequired)
        {
            SetTextCallback d = new SetTextCallback(SendOK_SetText);
            this.Invoke(d, new object[] { text });
        }
        else
        {
            if (okstate == 1)
```

```
            {
            //    textBox2.Text = "";
            //    textBox2.Text = text + sipifmatch;
             //   th.Abort();
            }
            else
            {
                msg = "";
            }
        }
    }


    private void button1_Click(object sender, EventArgs e)
    {
        if (comboBox1.SelectedIndex >= 0)
        {
            int timeinterval = Convert.ToInt32(comboBox1.Text);
            myTimer.Interval = timeinterval;
            myTimer.Tick += new EventHandler(Timer_Tick);
        }
        else
        {
            return;
        }
    }

    private void Timer_Tick(object sender, EventArgs eArgs)
    {

        if (state == 0)
        {
            String toAddress = textBox1.Text.ToString();
            this.publish(locationint, toAddress);
        }
        else
        {
            state = 3;
            String toAddress = textBox1.Text.ToString();
            this.publish(locationint, toAddress);
        }
        if (locationint <= 2)
        {
            locationint++;
        }
        else
        {
            locationint = 0;
        }
    }
```

```
private void publish(int i, string toAddress)
{

    String strHostName = System.Net.Dns.GetHostName();
    IPHostEntry ipEntry = System.Net.Dns.GetHostEntry(strHostName);
    IPAddress[] addr = ipEntry.AddressList;
    String myAddress = addr[addr.Length - 1].ToString();

    String branch = CreateRandom(7);
    String fromTag = CreateRandom(4);
    cSeq = "1";
    callID = CreateRandom(10);
    callID += System.DateTime.Now.ToString();
    String tupleid = CreateRandom(6);

    String Pidf = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
    "<presence xmlns=\"urn:ietf:params:xml:ns:pidf\" " +
        "xmlns:location=" +
        "\"http://it.kth.se/~moze/schemas/mohammad.xsd\"" +
    "entity=\"sip:ccsleft@" + toAddress + "\">" +
    "<tuple id=\"" + tupleid + "\">" +
    "<status>" +
    "<basic>open</basic>" +
    "<location> <description>" + location[i] +
    "</description> <room>" + "</room>" +
    "<floor>" +  "</floor>" + "<coordinates>" +
    "<latitude>" + "</latitude>" + "<longtitude>" + "</longtitude>" +
    "</coordinates>" + "</location>" +
    "</status>" +
    "<note>" + "</note>" +
    "<contact priority=\"0.8\">" + myAddress + "</contact>" +
    "</tuple>" +
    "</presence>" + "\r\n\r\n";

    String Pidf_length = Pidf.Length.ToString();
    if (state == 0)
    {
        string SipHeader =
        "PUBLISH " + "sip:ccsleft@" + textBox1.Text + " SIP/2.0" + "\r\n" +
        "Via: " + "SIP/2.0/UDP " + myAddress +
        ":5060;branch=z9hG4bK" + branch + "\r\n" +
        "To: <sip:ccsleft@" + toAddress + ">" + "\r\n" +
        "From: <sip:ccsleft@" + toAddress + ">;tag=" + fromTag + "\r\n" +
        "Call-ID: " + callID + "@" + myAddress + "\r\n" +
        "CSeq: " + cSeq + " PUBLISH" + "\r\n" +
        "Max-Forwards: 70" + "\r\n" +
        "Expires: 300" + "\r\n" +
        "Event: location" + "\r\n" +
```

```
        "Content-Type: application/pidf+xml" + "\r\n" +
        "Content-Length: " + Pidf_length + "\r\n\r\n";

        SipHeader += Pidf;
        textBox2.Text = "";
        textBox2.Text = location [i] + SipHeader;
        byte[] data = Encoding.ASCII.GetBytes(SipHeader);
        aserver =
        new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
        aserver.SendTo(data, data.Length, SocketFlags.None, serverip);
        aserver.Close();
        state = 1;
        okstate = 0;
    }
    if (state == 2 && (okstate == 1))
    {
        string SipHeader =
        "PUBLISH " + "sip:ccsleft@" + textBox1.Text
        + " SIP/2.0" + "\r\n" +
        "Via: " + "SIP/2.0/UDP " + myAddress +
        ":5060;branch=z9hG4bK" + branch + "\r\n" +
        "To: <sip:ccsleft@" + toAddress + ">" + "\r\n" +
        "From: <sip:ccsleft@" + toAddress + ">;tag=" + fromTag + "\r\n" +
        "Call-ID: " + callID + "@" + myAddress + "\r\n" +
        "CSeq: " + cSeq + " PUBLISH" + "\r\n" +
        "Max-Forwards: 70" + "\r\n" +
        "SIP-If-Match: " + sipifmatch + "\r\n" +
        "Expires: 300" + "\r\n" +
        "Event: location" + "\r\n" +
        "Content-Length: 0" + "\r\n\r\n";

        SipHeader += Pidf;
        textBox2.Text = "";
        textBox2.Text = location[i] + SipHeader;
        byte[] data = Encoding.ASCII.GetBytes(SipHeader);
        aserver =
        new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
        aserver.SendTo(data, data.Length, SocketFlags.None, serverip);
        aserver.Close();
        state = 1;
        okstate = 0;
    }
    if ((state == 3) && (okstate == 1))
    {
        string SipHeader =
        "PUBLISH " + "sip:ccsleft@" + textBox1.Text + " SIP/2.0" + "\r\n" +
        "Via: " + "SIP/2.0/UDP " + myAddress +
        ":5060;branch=z9hG4bK" + branch + "\r\n" +
```

```
            "To: <sip:ccsleft@" + toAddress + ">" + "\r\n" +
            "From: <sip:ccsleft@" + toAddress + ">;tag=" + fromTag + "\r\n" +
            "Call-ID: " + callID + "@" + myAddress + "\r\n" +
            "CSeq: " + cSeq + " PUBLISH" + "\r\n" +
            "Max-Forwards: 70" + "\r\n" +
            "SIP-If-Match: " + sipifmatch + "\r\n" +
            "Expires: 300" + "\r\n" +
            "Event: location" + "\r\n" +
            "Content-Type: application/pidf+xml" + "\r\n" +
            "Content-Length: " + Pidf_length + "\r\n\r\n";

            SipHeader += Pidf;
            textBox2.Text = "";
            textBox2.Text = location[i] + SipHeader;
            byte[] data = Encoding.ASCII.GetBytes(SipHeader);
            aserver =
            new Socket(AddressFamily.InterNetwork,
            SocketType.Dgram, ProtocolType.Udp);
            aserver.SendTo(data, data.Length, SocketFlags.None, serverip);
            aserver.Close();
            //state = 1;
            //okstate = 0;
        }
    }

    public static string CreateRandom(int Length)
    {
        string _allowedChars =
        "abcdefghijkmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
        Random randNum = new Random();
        char[] chars = new char[Length];
        int allowedCharCount = _allowedChars.Length;

        for (int i = 0; i < Length; i++)
        {
            chars[i] =
        _allowedChars[(int)((_allowedChars.Length) * randNum.NextDouble())];
        }

        return new string(chars);
    }

    private void button2_Click(object sender, EventArgs e)
    {
        this.receiveUdp.Close();
        this.th.Abort();
        if (state == 1)
        {
            this.aserver.Close();
        }
```

```
            System.Windows.Forms.Application.Exit();
        }

    }
}
```

Appendix E

Source code of the location_based reminder

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;

namespace location_based_reminder {
    public partial class Form1 : Form
    {
        private Context_collector context_collector;
        private Context_outlook context_outlook;
        private bool context_requested = false;
        private string location = "";
        private int count = 0;

        public Form1()
        {
            InitializeComponent();
            timer1.Enabled = false;
        }

        private void button1_Click(object sender, EventArgs e)
        {

            if (context_requested)
            {
                context_collector.stop_context_collector();
            }
```

```
            string serveraddr = textBox1.Text.ToString();
            context_collector =
            new Context_collector("location", serveraddr);
            context_requested = true;
            timer1.Enabled = true;

        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (context_requested)
            {
                context_collector.stop_context_collector();
            }
            System.Windows.Forms.Application.Exit();
        }

        private void timer1_Tick(object sender, EventArgs eArgs)
        {
            string temp = context_collector.get_context();
            this.textBox2.Text = location;
            if ((location != temp) && (temp != ""))
            {
                location = temp;
                count++;
                context_outlook = new Context_outlook(location);
                textBox3.Text = count.ToString();
            }
            location = temp;
        }
    }
}
```

Appendix F

Source code of the room booking viewer

This is a demo application that use the fixed update for room occupancy, which here we set the number of people in the room to be 3, instead of parsing it from the context_collector class. The reason for this is that as a demo to proof the idea, the room occupancy update means are still under discussion.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.IO;
using System.Xml;

namespace room_booker {
    public partial class Form1 : Form
    {
        private DataTable datatableA;
        private string xmldata;
        public string xmlvalue;
        string[] roomnamexml = new string[4];
        string[] starttimexml = new string[4];
        string[] endtimexml = new string[4];
        string[] roomindexxml = new string[4];
        string[] shortdescxml = new string[4];
        string[] bookingIDxml = new string[4];
        string[] entityxml = new string[4];
        string[] descriptionxml = new string[4];
```

```
string[] bookedbyxml = new string[4];
int state = 0;

public Form1()
{
    InitializeComponent();

}

private DataTable CreateDataTableA()
{
    DataTable aTable = new DataTable("A");
    DataColumn dtCol;
    DataRow dtRow;
    // Create ID column and add to the DataTable.
    dtCol = new DataColumn();
    dtCol.DataType = System.Type.GetType("System.String");
    dtCol.ColumnName = "RoomID";
    dtCol.AutoIncrement = false;
    dtCol.Caption = "RoomID";
    dtCol.ReadOnly = false;
    dtCol.Unique = true;
    // Add the column to the DataColumnCollection.
    aTable.Columns.Add(dtCol);
    dtCol = new DataColumn();
    dtCol.DataType = System.Type.GetType("System.String");
    dtCol.ColumnName = "People";
    dtCol.AutoIncrement = false;
    dtCol.Caption = "People";
    dtCol.ReadOnly = false;
    dtCol.Unique = false;
    aTable.Columns.Add(dtCol);
    // Create Name column and add to the table
    dtCol = new DataColumn();
    dtCol.DataType = System.Type.GetType("System.String");
    dtCol.ColumnName = "Status";
    dtCol.AutoIncrement = false;
    dtCol.Caption = "Status";
    dtCol.ReadOnly = false;
    dtCol.Unique = false;
    aTable.Columns.Add(dtCol);
    // Create Name column and add to the table
    dtCol = new DataColumn();
    dtCol.DataType = System.Type.GetType("System.String");
    dtCol.ColumnName = "Start";
    dtCol.AutoIncrement = false;
    dtCol.Caption = "Start";
    dtCol.ReadOnly = false;
    dtCol.Unique = false;
    aTable.Columns.Add(dtCol);
```

```
        // Create Name column and add to the table
        dtCol = new DataColumn();
        dtCol.DataType = System.Type.GetType("System.String");
        dtCol.ColumnName = "End";
        dtCol.AutoIncrement = false;
        dtCol.Caption = "End";
        dtCol.ReadOnly = false;
        dtCol.Unique = false;
        aTable.Columns.Add(dtCol);
        // Create three rows to the table
        dtRow = aTable.NewRow();
        dtRow["RoomID"] = "Motala";
        dtRow["People"] = "0";
        dtRow["Status"] = "free";
        dtRow["Start"] = "";
        dtRow["End"] = "";
        aTable.Rows.Add(dtRow);
        dtRow = aTable.NewRow();
        dtRow["RoomID"] = "Grimeton";
        dtRow["People"] = "0";
        dtRow["Status"] = "free";
        dtRow["Start"] = "";
        dtRow["End"] = "";
        aTable.Rows.Add(dtRow);
        dtRow = aTable.NewRow();
        dtRow["RoomID"] = "Horby";
        dtRow["People"] = "0";
        dtRow["Status"] = "free";
        dtRow["Start"] = "";
        dtRow["End"] = "";
        aTable.Rows.Add(dtRow);
        dtRow = aTable.NewRow();
        dtRow["RoomID"] = "Open Area";
        dtRow["People"] = "0";
        dtRow["Status"] = "free";
        dtRow["Start"] = "";
        dtRow["End"] = "";
        aTable.Rows.Add(dtRow);
        dtRow = aTable.NewRow();
        dtRow["RoomID"] = "Aloha";
        dtRow["People"] = "0";
        dtRow["Status"] = "free";
        dtRow["Start"] = "";
        dtRow["End"] = "";
        aTable.Rows.Add(dtRow);
        return aTable;
    }

    private void gethttp()
    {
```

```
            WebRequest request =
      WebRequest.Create(
      "http://www.cos.ict.kth.se/cgi-bin/cos-rooms-booking-annotated");
         // request.Proxy = null;

          // If required by the server, set the credentials.
          request.Credentials = CredentialCache.DefaultCredentials;

          // Get the response.
          HttpWebResponse response = (HttpWebResponse)request.GetResponse();

          // Get the stream containing content returned by the server.
          Stream dataStream = response.GetResponseStream();

          // Open the stream using a StreamReader for easy access.
          StreamReader reader = new StreamReader(dataStream);

          string responseFromServer = reader.ReadToEnd();
          int index = responseFromServer.IndexOf("<head>");
          string subresponseFromServer =
          responseFromServer.Substring(index, responseFromServer.Length - index);

          xmldata = "<?xml version = \"1.0\"?>" + "\n" + "<html>" + "\n";
           xmldata += subresponseFromServer;
/*          string xmlfile =
  "<head>" + "\n" +
  "<TITLE>Room bookings at CoS</TITLE>" + "\n" +
  "</head>" + "\n" +
  "<body>" + "\n" +
  "<roombookings>" + "\n" +
  "<roombooking>" + "\n" +
  "<roomindex>136855289</roomindex>" + "\n" +
  "<roomname>Grimeton</roomname>" + "\n" +
  "<starttime>1193914800</starttime>" + "\n" +
  "<endtime>1193922000</endtime>" + "\n" +
  "<shortdesc>Booking Grimeton</shortdesc>" + "\n" +
  "<bookingID>2910</bookingID>" + "\n" +
  "<entity>CoS</entity>" + "\n" +
  "<description>Master's thesis presentation</description>" + "\n" +
  "<bookedby>Yu Sun</bookedby>" + "\n" +
  "</roombooking>" + "\n" +
  "<roombooking>" + "\n" +
  "<roomindex>136855290</roomindex>" + "\n" +
  "<roomname>Aloha</roomname>" + "\n" +
  "<starttime>1193914800</starttime>" + "\n" +
  "<endtime>1193922000</endtime>" + "\n" +
  "<shortdesc>secondbooking</shortdesc>" + "\n" +
  "<bookingID>2911</bookingID>" + "\n" +
  "<entity>CoS</entity>" + "\n" +
  "<description>This is a booking of the room Aloha</description>" + "\n" +
```

```
"<bookedby>sunyu</bookedby>" + "\n" +
"</roombooking>" + "\n" +
"</roombookings>" + "\n" +
"</body>" + "\n" +
"</html>" + "\n";
      xmldata += xmlfile;
*/
  }

  private void button1_Click(object sender, EventArgs e)
  {
      datatableA = CreateDataTableA();
      dataGrid1.DataSource = datatableA;
      gethttp();
      state = 1;
  }

  private static void ModifyRow(DataTable myDataTable,
  string roomid, string newnumberofpeople,
  string status, string starts, string ends)
  {
      // step 1: set the PrimaryKey property for the DataTable object
      DataColumn[] myPrimaryKey = new DataColumn[1];
      myPrimaryKey[0] = myDataTable.Columns["RoomID"];
      myDataTable.PrimaryKey = myPrimaryKey;

      // step 2: use the Find() method to locate the DataRow
      // in the DataTable using the primary key value
      DataRow myEditDataRow = myDataTable.Rows.Find(roomid);

      // step 3: change the column values
      myEditDataRow["People"] = newnumberofpeople;
      myEditDataRow["Status"] = status;
      myEditDataRow["Start"] = starts;
      myEditDataRow["End"] = ends;

      // step 4: use the AcceptChanges() method of the DataTable to commit
      // the changes
      myDataTable.AcceptChanges();
  }

  private void button2_Click(object sender, EventArgs e)
  {
      Application.Exit();
  }

   private void button3_Click(object sender, EventArgs e)
  {
      if (state != 1)
      {
```

```
            return;
        }
        else
        {
            int number_of_booking = getxml(xmldata);

            for (int i = 0; i < number_of_booking; i++)
            {
                long dbstart = System.Int32.Parse(starttimexml[i]);
                long dbend = System.Int32.Parse(endtimexml[i]);
                DateTime dtstart =
                new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
                DateTime dtend = dtstart;
                string starttime =
                dtstart.ToLocalTime().AddSeconds(dbstart).ToString();
                string endtime =
                dtstart.ToLocalTime().AddSeconds(dbend).ToString();

                int index = starttime.IndexOf(' ');
                int shorttimelength_start =
                starttime.IndexOf(':') - index - 1;
                int shorttimelength_end =
                endtime.IndexOf(':') - index - 1;
                string  starttime_short =
                starttime.Substring(index + 1, shorttimelength_start)
                     + starttime.Substring((starttime.Length - 2), 2);

                string endtime_short =
                endtime.Substring(index + 1, shorttimelength_end)
                     + endtime.Substring((endtime.Length - 2), 2);

                ModifyRow(datatableA, roomnamexml[i],
                "10", "booked", starttime_short, endtime_short);
            }
            dataGrid1.DataSource = datatableA;
        }
    }

    private int getxml(string dataFeed)
    {
        XmlReader ContextReaderAgent =
        new XmlTextReader(new System.IO.StringReader(dataFeed));
        string[] context = new string[9];
        int count = 0;

        while (ContextReaderAgent.Read())
        {
            if (ContextReaderAgent.NodeType == XmlNodeType.Element)
            {
                if (ContextReaderAgent.LocalName.Equals("roomname"))
```

```
                {
                    context[0] = ContextReaderAgent.ReadString();
                    roomnamexml[count] = context[0];
                }
                else if (ContextReaderAgent.LocalName.Equals("starttime"))
                {
                    context[1] = ContextReaderAgent.ReadString();
                    starttimexml[count] = context[1];
                }
                else if (ContextReaderAgent.LocalName.Equals("endtime"))
                {
                    context[2] = ContextReaderAgent.ReadString();
                    endtimexml[count] = context[2];
                }
                else if (ContextReaderAgent.LocalName.Equals("roomindex"))
                {
                    context[3] = ContextReaderAgent.ReadString();
                    roomindexxml[count] = context[3];
                }
                else if (ContextReaderAgent.LocalName.Equals("shortdesc"))
                {
                    context[4] = ContextReaderAgent.ReadString();
                    shortdescxml[count] = context[4];
                }
                else if (ContextReaderAgent.LocalName.Equals("bookingID"))
                {
                    context[5] = ContextReaderAgent.ReadString();
                    bookingIDxml[count] = context[5];
                }
                else if (ContextReaderAgent.LocalName.Equals("entity"))
                {
                    context[6] = ContextReaderAgent.ReadString();
                    entityxml[count] = context[6];
                }
                else if (ContextReaderAgent.LocalName.Equals("description"))
                {
                    context[7] = ContextReaderAgent.ReadString();
                    descriptionxml[count] = context[7];
                }
                else if (ContextReaderAgent.LocalName.Equals("bookedby"))
                {
                    context[8] = ContextReaderAgent.ReadString();
                    bookedbyxml[count] = context[8];
                    count++;
                }

            }
        }

        return count;
```

```
}

private void button4_Click(object sender, EventArgs e)
{
    DataRow row =
    ((DataTable)dataGrid1.DataSource).Rows[dataGrid1.CurrentRowIndex];
    string selected_room = row.ItemArray[0].ToString();
    textBox1.Text = selected_room;
    string details;
    int booking_detail_index = 1000;

    for(int i = 0; i< 4; i++)
    {
        if (roomnamexml[i] == selected_room)
        {
            booking_detail_index = i;
        }
    }

    if (booking_detail_index != 1000)
    {
        long dbstart =
        System.Int32.Parse(starttimexml[booking_detail_index]);
        long dbend =
        System.Int32.Parse(endtimexml[booking_detail_index]);
        DateTime dtstart =
        new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
        DateTime dtend = dtstart;
        string starttime =
        dtstart.ToLocalTime().AddSeconds(dbstart).ToString();
        string endtime =
        dtstart.ToLocalTime().AddSeconds(dbend).ToString();

        details =
        "Roomindex: "
        + roomindexxml[booking_detail_index] + "\r\n" +
        "RoomID: "
        + roomnamexml[booking_detail_index] + "\r\n" +
        "Start time: " + starttime + "\r\n" +
        "End time: " + endtime + "\r\n" +
        "Short description: "
        + shortdescxml[booking_detail_index] + "\r\n" +
        "BookingID: "
        + bookingIDxml[booking_detail_index] + "\r\n" +
        "Entity: "
        + entityxml[booking_detail_index] + "\r\n" +
        "Description: "
        + descriptionxml[booking_detail_index] + "\r\n" +
        "Booked by: "
        + bookedbyxml[booking_detail_index] + "\r\n\r\n";
```

```
                textBox1.Text = details;
            }
        }
    }
}
```

COS/CCS 2007-28

www.kth.se