Université d'Ottawa • University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES
ET POSTDOCTORALES

FACULTY OF GRADUATE AND
POSTDOCTORAL STUDIES

Prasanna GANESAN
AUTEUR DE LA THÈSE - AUTHOR OF THESIS

Master of Computer Science
GRADE - DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Context Information Management Using Agents

A. Karmouch
DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

S. Dandamudi

T. Yeap

J.-M. De Koninck, Ph.D.
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

SIGNATURE

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Context Information Management Using Agents

By

Prasanna Ganesan, B.E.

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
Department of Computer Science
School of Information Technology and Engineering
Faculty of Engineering

University of Ottawa
Ottawa, ON, Canada

Your file Votre référence
ISBN: 0-494-01475-X
Our file Notre référence
ISBN: 0-494-01475-X

Abstract

New trends and emerging technologies have realized the omnipresence of electronic goods in the market. Computers are becoming a part of day-to-day activities and its presence can be felt in every walk of life. With the constant increase in processing power of computers, researchers are continually working towards using that power meanwhile hiding computing capabilities in the background. Ubiquitous and invisible computing thereby gains more attention from researchers all around the globe and several leading companies have provided their input in this regard. Context awareness is an approach towards realizing ubiquitous computing.

The thesis is about an effort taken towards achieving context awareness by efficiently managing context information. The model discussed, is developed in MMARL (Multimedia and Mobile Agent Research Laboratory) at University of Ottawa. The ideal setting for the context management model is a sensor-rich environment made up of stationary and mobile objects like people and services. Context of these entities forms the central theme of the proposed context management model and all the activities are targeted towards managing context data and making use of them in automatic task execution.

The management of context spans from defining context attributes to triggering of actions resulting from a context state change. The design of the model is targeted towards creating a context aware environment for an adhoc users/services collaboration system. Contextual data are grouped and stored as profiles and database tuples depending on context themselves. The context management model gathers profile and preferences of mobile users and services and monitors changes that the entities undergo. These changes are reflected as actions on the related entities depending on the context change. The thesis proposes a design, which enables creating relationships between context attributes and deriving semantics out of them to describe a situation. Once the situation is resolved, the model identifies the tasks to be carried out in such situation and triggers relevant actions.

The scope of context management model is restricted to entities that are present in a physical room. There can be more than one such room in several geographical

locations. The context of the entities is captured through sensors and its behavior is controlled using the captured context based on the policy and preferences of related entities. An interesting aspect of the model is that since it is agent based, communications between similar agents of other models become easier. Agent model subsequently enhances the scalability of the design through the capability of incorporating external features and hence extensions to this model can be done with least effort.

A prototype model was developed for this architecture using Java and FIPA-OS as the agent platform. The various features of this prototype are discussed and the performance issues that help in gaining further insight about the model are discussed. The prototype implemented was resourceful in obtaining practical knowledge and experience in developing a context-aware model and the applications in an ad hoc environment.

Acknowledgements

I would like to thank my supervisor Dr. Ahmed Karmouch for his valuable guidance, motivation, and support. I would like to extend warm gratitude to my colleagues at Multimedia and Mobile Agent Research Laboratory, University of Ottawa for their suggestions and cooperation. I also thank Tom Gray, Ramiro Liscano, Roger Impey and Mohammed Ahmed from proving valuable suggestions towards this thesis. I extend my thanks to NSERC, NRC and MITEL for their precious suggestions and monetary support that helped in successful completion of masters' program. I wish to thank my parents and my family for providing moral support and appreciation throughout my period of study

Contents

List of Figures

List of Abbreviations

CMS	Context Management System
FIPA-OS	Foundation of Intelligent Physical Agents – Open Source
ACL	Agent Communication Language
ACC	Agent Communication Channel
MTS & MTP	Message Transport Service and Protocol
HTTP	Hyper Text Transport Protocol
XML	Extensible Markup Language
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
PDF	Portable Document Format
ADB	Agent Deployment Block
MP3	MPEG audio encoding layer III
QH	Query Handler
DSB	Data Storage Block
MANET	Mobile Adhoc NETwork

Chapter 1

Introduction

Recently considerable efforts are being put on smart automation of various jobs in the daily life. Most of the research work is in one way or other targeted towards solving the user concerns with the least amount of user intervention. The basic ingredients for this intelligence are a galore of miniscule data that adds to the description of various related entities and formation of relationships between them. In other words, it is deriving a semantic from the information that is in hand. Technically these small data about an entity make up the *context information* of an entity. When an application is able to identify and capture these data and trigger appropriate action, then it can be called as a context-aware application.

Context-awareness forms the basis of artificial intelligence where an application constantly learns and adapts to its environment. A solid information base forms the basis of a context-aware system. The thesis proposes one such effort taken towards managing the context data using software agents and providing them to the higher end applications.

A general introduction to agents is given and some agent-based models of similar genre are discussed. Following that is a detailed description on Context Management and some of the existing models. And finally the implementations and the issues that arose consequently are discussed.

1.1 Context Definition

The term "*Context*" is defined in Merriam-Webster Dictionary as "*interrelated conditions in which something exists or occurs*". It means that the context of an entity encompasses all the elements that this entity influences or is influenced by. There have been plenty of definitions that are given to "*Context*" in the field of computing. Schilit, Adams et.al [16] define context to be a continuously changing environment. The environment is classified as *Computing* (Processors, i/o devices, etc), *User* (Location, nearby users), *and Physical* (Lighting, temperature, noise level, .etc). Another more detailed definition was given by Dey [48], which says,

> *"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."*

In general, context can be defined with respect to a particular object as,

> *"Context of an entity is any piece of descriptive information that may directly influence the state or behavior of that entity"*

Kotz and Chen [1] classified context aware computing into two broad categories- Active and Passive Context.

Active Context: The application will automatically adapt to the changes in these context variables by changing the behavior of the application

Passive Context: The application just presents the new or changing context to the user and makes it persistent.

Thus from the various interpretations of context it can be seen that an entity may be associated with a galore of contextual information. Consequently generalization of context encompassing all the parameters proves to be a time consuming and ineffective approach.

The three important aspects of context according to B.Schilit [16] are *where you are, who you are with and what resources are nearby.* But the time context has also proved equally important in most of the context-aware applications.

1.2 Context-aware Computing

This section details context-aware computing and the various nomenclatures that are associated with it. Context-awareness is basically manipulating and using context information to solve problems. A Context aware application can thus generally be defined as,

An application that is able to identify sense and use the context to perform or trigger relevant actions.

There have been various efforts made towards classification of context aware applications. Pascoe [15] classified it into three categories,

Contextual sensing, in which the application is targeted towards obtaining the user's current location,

Contextual adaptation, in which the application acts according to a particular set of context attributes,

Contextual resource discovery, in which the application finds services relating to current context, and

Contextual augmentation is augmenting or tagging a particular context attribute with an action or a message.

A deeper look at the similarities in the last two categories which is resource discovery and augmentation, reveal that both of them are aimed at triggering an action according to a context attribute.

Context-awareness forms the basis of *Pervasive Computing, Ubiquitous Computing or Human Computer Interaction.* Ubiquitous/Pervasive computing is making computers available seamlessly throughout the environment of mobile users and computers in preserving the privacy and the preferences of the mobile entities [2]. The ultimate goal of ubiquitous computing is to make computers invisible to the user while retaining the power of computing.

The *Aura Project* [13] from Carnegie Mellon University is aimed at providing invisible computing to its users despite their location. This project embraces wearable, Handheld, Desktop and Infrastructure computers. Another similar project is *"Oxygen"* [14] from MIT, which aims at creating ubiquitous human-centered environment where

the computer is made to understand normal human activities and respond to the user accordingly.

1.2.1 Context-Awareness in Applications

Context-awareness is a desirable feature to most of the general applications as it makes the application adaptive and efficient. Context-awareness reduces human – interference to a great extent and automates tasks. It can also be found that most of the work in context-aware computing is limited to research and not products. There are several issues and complications involved in creating context awareness. Pascoe et.al [17] gives a good description of the issues and taxonomy that are involved in creating context-awareness stating that the context-aware applications are resource hungry, require high development costs and the computing environments that these applications span is large and diverse. The more functionality the application provides the more context attributes are considered in the application, and thus the complexity of the application increases.

Context-awareness can also be visualized as an add-on to an existing application, where some processes in the application are automated or new decision-making processes are added. A simple example in this case would be the Microsoft Word application where the pop-up helper automatically provides with the contextual help menu that the user might be requiring from the sentences typed or mouse clicked. The scope of user's context here is restricted to the keys pressed, idle time etc. The helper can be considered as an additional feature that can be obtained by making an application "Context-aware".

1.3 Ad-hoc Networks

An ad-hoc network [38,39] is formed dynamically by mobile devices and is managed by the nodes that enter and leave this network. Thus an ad-hoc network may have any network topology, and may or may not have a gateway to any particular fixed network. The users may bring their mobile devices and get connected to the network without any prior configuration. The ad-hoc network can be visualized as a wireless LAN where the users bring in various types of wireless devices like a palm-top or a notebook. The devices once turned on get spontaneously connected and can communicate with other connected devices or use the services that are in this network. Due to this spontaneous and dynamic nature of the ad hoc network, it proves to be an interesting and challenging direction for researchers.

1.3.1 Need for Context-Awareness

For each node to route the packets it has to be aware of the network properties like the number of nodes in the network, the bandwidth of the network, the resources that are available in the network. Each node has to be continuously updated with the composition of the network like the services, the users and network properties. This information is useful for nodes to filter the information and route the packets. In an ad-hoc network a node has to choose from received data, those that are relevant to it and decide on forwarding or destroying other data packets.

More than just routing, context awareness is also used to create an ad-hoc network. For example, from the context information of a group of people, an ad-hoc network can automatically be started and proceed for a particular length of time or

operate indefinitely until an external event is triggered. It can also be a conferencing session between multiple users that may involve resources like video/audio. Context-awareness helps in discovering those services or nearby resources.

1.3.2 Challenges in Building Context-Aware Models

This section details the problems and challenges that arise when trying to create context aware models in adhoc environments. There have already been some previous efforts [18][19] to address this problem of communication in ad-hoc networks through context aware computing.

The primary challenge lies in capturing or identifying the various types of devices involved in the network. The node has to gather context from various types of sensors either by equipping itself with all necessary mechanisms or communicate with a common context-aware system responsible for that location. Moreover this information has to be refreshed frequently to monitor the changes that the network or the other nodes may undergo.

Another challenge that we may arise at this juncture is identifying the nodes and carrying out communication with them. The description and capabilities of every node must be understood by all the other nodes so as to enable discovery and sharing resources. All the nodes of the network, irrespective of the device type, should support a common subset of communication protocol.

The more common of the problems that the researchers have been striving to tackle is the privacy and the security issues of the nodes in the network. Since it is a spontaneous and unplanned network, legitimate nodes must be protected from attacks by

anonymous devices. The privacy and preferences of mobile terminals should be secured and exploitation of the network resources should be checked. It should be carried out in way that does not tamper the flexibility and dynamic nature of the network. This is one trade-off that has to be carefully dealt with while enforcing security constraints in an ad-hoc network. Adding context-awareness to the ad-hoc network may further expose the personal information to the network demanding more focus on security.

1.4 Thesis Contributions

The results of the thesis are -

(i) *Conception and implementation of an agent-based context-aware system model*

The thesis proposes a design that uses software agents to manage context data and interpret into forms understood by third party applications. The proposed architecture assumes that all the participating devices in the design support or capable of supporting agents.

(ii) *Conception and realization of context-data manipulation during runtime*

A novel way to handle or manage context data is being described in the thesis in which the context data processing is subject to change depending upon the situation described what we refer to as a Context Object.

(iii) *Context archiving and adaptive learning*

As a next step towards use of context, derivations are done on the stored context. The architecture proposed incorporates archiving mechanism which provides stored context information on all related entities and makes them available through querying mechanisms for the upper layers to aid them in making relevant decisions.

1.5 Frequently Used Keywords and Definitions

Context: Context of an entity is any piece of descriptive information that may directly influence the state or behavior of that entity

Context – Aware: An application is said to be context aware when it is capable of identifying the context information that directly or indirectly influence the performance of application.

Location – Aware: Applications that require the location information of the entities involved to carry out its tasks.

Ubiquitous Computing: Making many computers available throughout the physical environment, while hiding them effectively from the user

Agents: A software program that is authorized to act for another in carrying out actions on behalf of the client like decision making, committing resources and performing tasks.

Ad hoc Network: "Ad hoc" in Latin means "for this". An Ad hoc or spontaneous network is local are a network with wireless or temporary plug-in connections, in which some of the devices are a part of the network only for the duration of a communication session.

Mobile Node: A mobile node is a network -connected device whose location and point of attachment to the network may frequently be changed.

RFID: Radio Frequency Identification is a technology of using electromagnetic or electrostatic coupling in the radio frequency portion of electromagnetic spectrum to uniquely identify an object.

GPS: The GPS (Global Positioning System) is a constellation of 24 well-spaced satellites that orbit the Earth and make it possible for people with ground receivers to pinpoint their geographic location. The location accuracy is anywhere from 100 to 10 meters for most equipment

Policies: A policy is a formal set of statements that define how the network's resources are to be allocated among its clients. Clients can be individual users, departments, host computers, or applications.

XML: XML (Extensible Markup Language) is a flexible way to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere.

CC/PP: CC/PP stands for *Composite Capabilities/Preferences Profiles*, and is a way to specify what exactly a user agent (web browser) is capable of doing. This allows for sophisticated content negotiation techniques between web servers and clients, to produce optimized XML-based markup for display and use on a wide variety of web user agents.

RDF: RDF is a standard way of expressing metadata, specifically resources on the Web, although it can be used to represent any kind of structured data. It is based on "triples", where each triple expresses the fact that an object **O** has attribute **A** with value **V**, written as **A(O,V)**.

Triangulation: Triangulation is a process by which the location of an object can be determined by measuring either the radial distance, or the direction, of the received signal from two or three different points. This is used in location tracking systems to pin point the exact location of a user in a geographical area.

1.6 Thesis Outline

The thesis presents an architectural model to support context management in a mobile ad hoc environment. Chapter two deals with the background knowledge that is

required to understand the design and working of the context management architecture. It describes the fundamental idea of context and context-awareness and why it is one of the most sought research area. It also introduces agent technology and their participation in creating context awareness and some of the service discovery protocols that aid in this model and the usage of context in the discovery process.

Chapter three gets into the central idea of the thesis, which is context detection and management. It also discusses the techniques and tools described in chapter two at various phases in the detection and management process. Chapter four forms the core of the thesis, which describes the context management model with detailed explanation on each and every block of the model. As the penultimate chapter of the thesis, chapter five presents the implementation of a prototype of the context management model. Chapter 5 also features some of the snapshots that were taken during the execution of the model and some interesting code snippets. The performance of the prototype was also charted in this chapter to evaluate the model and the knowledge that was gained during the process of building the context management model.

The final chapter ends with the contributions of this thesis and possible future research directions to this work.

Chapter 2

Background and Related Work

This chapter is intended to explain some of the techniques that are related to the subject of this thesis. Some of the existing techniques and groundwork that aid in creating a context management model are discussed. The different ways of capturing context through sensors and how to represent the captured context so that various modules can use the captured context are analyzed. This chapter also discusses service discovery and some of the existing protocols involved in it. Further more a brief analysis of how context can affect service discovery process and improve performance is being done. Following that is a quick overview of agents and some of agent platforms showing how mobile agents are used in ad-hoc networks. Finally some of the existing projects that aim at creating context-aware environment are discussed.

2.1 Context Detection and Representation

The preliminary step in every context-aware system is the detection or capturing of elements or objects in the environment. This section presents some of the sensors that are used to capture entities. The entities in an ad hoc environment that are of interest are users and services. The users are identified through RF detectors and the services through network sensors. These sensors are discussed in detail in the subsequent sections.

2.1.1 Sensors

An ad-hoc environment may contain any number of users and different types of services. Identifying their presence is a key requirement to efficiently manage an ad-hoc network. The sensors can generally be classified into two categories – *Hardware sensors and Software sensors*. Those that require special hardware devices to sense objects are termed as hardware sensors. These sensors are used to capture physical parameters in the environment like temperature, pressure or presence of a physical object. The software sensors capture the context variables through software programs. Some of the context parameters that can be captured through software sensors are the device properties like terminal type, software or processes running in the device and the connectivity with the network.

2.1.1.1 Active Badges

Active badge [20] is a system designed to identify the location of the users in the geographical location, ideally an office environment. The system was designed and

prototyped between 1989 and 1992 by AT&T laboratories, Cambridge and Cambridge University Engineering Department. The tags based on infrared technology that transmits a beacon every 15 seconds. The beacon contains a unique code that identifies the tag that is transmitting this beacon. These beacons are picked up by readers or IR sensors connected together in a network. A master station that is connected to that network polls these readers and obtains the code to identify the tag that emitted the beacon. By assigning these unique tags to users in a geographical area, their presence can be monitored by this system. The largest single system is at Cambridge University Computer Laboratory, which uses over 200 badges and 300 sensors. They can be used for applications based on location-awareness like call forwarding or a tourist guide. The following figure shows an active badge that transmits beacons and a network sensor that picks up the beacons.

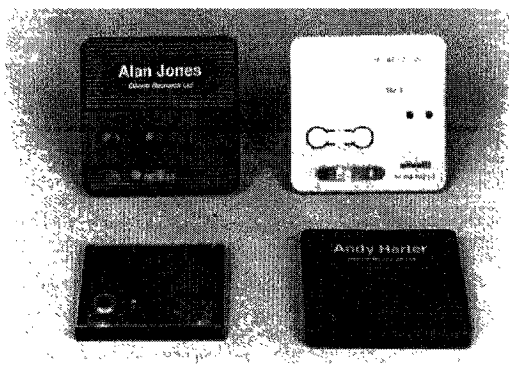

Figure 2.1: Active Badge-4 Different Models [20]

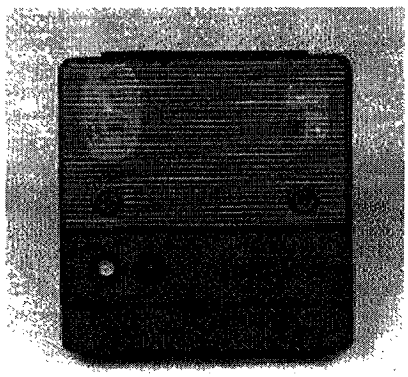

Figure 2.2 Network Sensor [20]

There are other similar indoor location sensors like Bats [22] and Cricket [23]. They can calculate the position of the tag and sensors through 2D lateration.

2.1.1.2 Piconets

Piconets [25] are low-power ad-hoc networks in which two or more devices recognize each other and communicate. The mobile devices that are equipped with piconet nodes can communicate with each other when they are in close proximity. *Piconet* is under development at Olivetti and ORL (Oracle Research Laboratory). When smart devices like alarms or calendars are equipped with piconet nodes, they can transmit their state over the piconet to other devices in the proximity. Piconet provides a limited range Communication channel. Therefore it can also act as a sensor that identifies other devices in close proximity. This identification can be used to trigger actions, and thus Piconet can be used for *context aware* applications. The following is a picture of the prototype Piconet node.

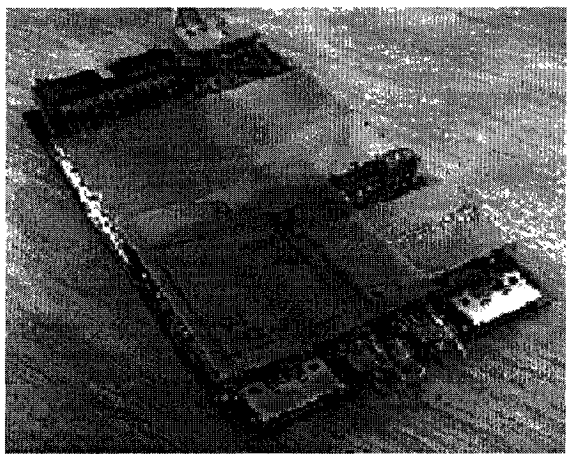

Figure 2.3 Prototype piconet node [25]

When this node is attached to mobile devices, they form a network when brought together in close proximity. In a mobile ad-hoc network, when a specific set of devices are brought together, then the appropriate applications running on the devices can be informed about the presence of the other proximate devices and could trigger an action. Piconets support a variety of devices from a simple binary switch to a notebook or a PDA. Piconet has a property of decentralized resource discovery where each piconet node describes its own capabilities or services. This feature makes it highly advantageous in an ad-hoc network.

2.1.1.3 GPS

GPS (Global Positioning System) [26] is a worldwide navigation system that is used to calculate the absolute position of the object on the globe. It is an outdoor positioning service, which calculates the position based on the 24 satellites and their specific ground points. This system can be used almost everywhere except inside

buildings, underwater, or in caves. This uses the technique of triangulation to calculate the position and accuracy is close to a meter.

GPS is mainly used to calculate the location of an object. The other application types that use GPS are navigation, tracking, mapping and timing. GPS is now commonly used in automobiles to find the location or a route to a particular destination.

2.1.2 Representing Context

The captured context has to be represented in ways that enable various modules to exchange and understand the context. One way is to encode context information into XML tags. This assures flexibility and diversity in data definitions. In further subsections a couple of representations using XML to effectively represent context information are discussed.

2.1.2.1 RDF and CC/PP

Resource Description Framework (RDF) [27] is one way of representing metadata proposed by the W3C (WWW Consortium). The idea of metadata started from Platform for Internet Content Selection (PICS) [28]. Metadata can be defined as "*information about data*". Here data can be considered as an object like a document or a system. RDF is used to represent that descriptive information understood by other RDF speaking communities. RDF follows the XML representation so that it is system independent and understandable. RDF represents a resource uniquely identified by a URI (Uniform Resource Identifier). Each of these resources or objects has a set of *properties*

or context attributes. A collection of these properties makes up the *description* of the object.

The following presents a simple example of RDF. The resource that will be described in this example is a person.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about = "http://www.uottawa.ca/students/11001">
    <s:name>John</s:name>
    <s:location>CBY</s:location>
    <s:Email>"john@site.uottawa.ca"</s:Email>
  </rdf:Description>
</rdf:RDF>
```

The above example shows a complete XML representation that provides description about a student named john with student number, email, and location.

CC/PP (Composite Capabilities/Preference Profiles) is used for content adaptation and negotiation. It is a RDF based profiler that contains a collection of description or capabilities of a device or service together with the preferences of the user. CC/PP can be thought of as a collection of RDF statements that describes the services and users. It is stored in form of tables and each table is a collection of RDF statements. The main purpose of CC/PP is negotiating content between the two parties to facilitate the identification of right kind of service. The RDF data in the CC/PP profile may originate

from various sources and merged together into a single CC/PP profile. The description of the device may be from the device manufacturer and the preferences from the user's personal assistant. The content server compares this document with stored documents to find the best match of service provider for this user.

2.1.2.2 conteXtML

ConteXtML [29] is simple XML based protocol for exchanging contextual information. All the conteXtML messages are grouped between <context> tags. This protocol has a set of pre-defined tags. The main task of this protocol is to enable client and server to send and receive context. They have specific set of attributes and values that are used for creating and managing communication sessions. Database queries can also be sent in this conteXtML message to obtain the context values.

2.1.2.3 Policies

Policies are rules that govern the behavior of entities with a specific domain. Policies [30] are generally applied in security – for restricting access, management – to assign rules participating entities, and conversational policies – to structure and carry out conversation between entities. There are several policy languages [31][32] that are aimed at formalizing the specification of policies so that they can be represented and interpreted by machines.

Policies can be considered as a tool for interpreting context, since the basic elements that form the policy statements are context attributes of one or more entities. For example, an organization might specify that,

"Fax service is available only between 8 AM to 8PM and be accessed only by users with grade level above A"

It can be noted here that this statement involves the context of the fax service, time context and user's context. The context changes of the entities should be constantly monitored to govern their behavior through policies.

2.2 Service Discovery

Service Discovery is the process of searching for new services and providing means of accessing the discovered services. The services can be both hardware and software services that are available in the network. The objective of service discovery is to identify the availability or unavailability of services and to provide comprehensive description on the services in the network when it is requested. The discovery can be centralized, in which there is a common SD server that constantly checks the network for service updates, or distributed, in which each node in the network is responsible for getting the service updates.

2.2.1 Existing Discovery Protocols

This section discusses a few existing protocols [33] for service discovery. A service discovery protocol provides a way for applications to discover services and identify their properties in a network. There are also two modes of action in these protocols. The *push* mode is wherein the service update is advertised to all the nodes in a network and in the *pull* mode the nodes query or ping the network to get service updates.

SLP (Service Location Protocol) [50] was developed by IETF (Internet Engineering Task Force) for TCP/IP networks and is a vendor independent standard. The architecture contains three parts- the service agents (SAs), the user agents (UAs), and the directory agent (DA). SAs and UAs represent services and users respectively and the DA maintains the list of users and services and controls their behavior.

Jini [51] is another Service Discovery protocol developed by Sun Microsystems that is based on java. It has a similar architecture as SLP and communication is carried out by exchanging serialized java objects. These objects can encapsulate the code for accessing the service and thus reduces the significant amount of communication. But since Jini is based on Java, other programming languages cannot use it. Jini is not suited for small devices with very less memory since it requires JVM to be installed and running.

Both Jini and SLP require the services to register itself with the lookup service and has a time stamp associated with it. It is called as *leasing concept* in which the service is bound to renew its registration before the expiry time to indicate its active state.

Salutation [52] is another protocol from the open industry Salutation Consortium. It operates in a similar way as SLP and Jini, but is defined above the transport layer, thus making it more technology independent. More examples of discovery protocols include Microsoft's Universal Plug and Play (UPnP) [49] and Bluetooth [53].

2.2.2 Using Context in Service Discovery

Having realized the concept of service discovery the next step is to analyze the role of context in service discovery process. All the service discovery protocols need a

trigger to carry out their tasks, which may be generated through a request from any node or a user in the network. This request can be automatically generated on behalf of the node by capturing and analyzing the node's context. A simple example below would state the importance of context in smart automated service discovery.

Assume that there are a few users connected to the network through their mobile devices like PDAs or notebooks. The personal assistant from one of the user's devices detects that there is a schedule for presentation and that all the required persons involved in the presentation are connected to the network. This agent may initiate a service discovery process to identify an appropriate projector to flash the presentation.

2.3 Agent Technology

This section presents a brief introduction on software agents, common definitions and some of the existing agent platforms. The differences between agent models and client server models and how agent models are better suited for context-aware applications are discussed.

2.3.1 Agent Definition

Agents are simple software programs designed to execute a special set of tasks independently or by collaborating with other agents. Agents can be created or destroyed dynamically and this adds to the flexibility of their usage. The agents are autonomous and carry out a specific set of tasks that it is programmed to do. The most interesting feature of an agent is that it is mobile, which means that it can be created at one location and executed in another location.

The following are some of the most commonly perceived definitions on agents.

The Maes Agent [3] *"Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed."*

"Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires." [4]

The latter is a more general definition of an agent. The most sought variant of agents is the mobile agent which performs the normal above said tasks and in addition it also has the capability to roam in a network carrying cargo along with it. Cargo here refers to the data that is required for the agent to perform its tasks.

An agent is generally associated with a task and is let loose in the network for it to complete the specified task. A simple agent will generally have a specified itenary or path which it traces in order to complete the task and in the end report the results to the owner. But a more intelligent [54] agent on the other hand may change its course on the fly depending upon its status at a particular node. These kinds of agents attract considerable research attention due to their vast applicability in various areas.

2.3.2 Agent Platforms

For an agent to move from one node or network to another and interact with foreign agents, the target platform must be able to recognize this agent and understand its

language. Therefore it is necessary that there is a common platform which recognizes agents and their semantics. So far the agent platforms are built as an application on the operating system.

The agent platforms have a general specification given by FIPA (Foundation for Intelligent Physical Agents) [5] or MASIF (Mobile Agent System Interoperability Facility) [6], which is the first mobile agent industry standard.

Two of the agent platforms that are more prevalent are Grasshopper [7] developed by GMD FOKUS and IKV++ GmbH and the other one is FIPA-OS.

Grasshopper is a mobile agent development and runtime platform developed over a distributed processing environment. Grasshopper achieves an integration of the traditional client/server paradigm and mobile agent technology. It is implemented in Java 2 and it conforms to MASIF and FIPA standards. Currently *Grasshopper* is widely used in international research projects in Europe. The core of Grasshopper consists of a *Communication Service* for all remote interactions between various distributed components of the agent platform, a *Registration Service* for each agent to know about the other currently registered or available agents, a *Management Service* to monitor and control the agents, a *Security Service* and *Persistence* to recover agents in case of a system crash. The Grasshopper consists of two kinds of security. They are,

External Security – To protect remote interaction between agents of remote agencies or locations

Internal Security – To check unauthorized use of resources in one platform by hosted agents and also provide protection from other agents in the same platform.

Grasshopper contains a FIPA add on to make it compatible with FIPA compliant agent platforms due to the increased acceptance of FIPA standard.

FIPA-OS is a component based toolkit developed in Java and is compliant with FIPA standards only. FIPA-OS is also used in some of the European projects like CRUMPET, SHUFFLE etc. FIPA-OS has another exclusive version for PDAs known as the Micro-FIPAOS developed at University of Helsinki. It uses IIOP, RMI, and ACL for communications but it does not support mobility. The latest release of the Standard FIPA-OS is version 2.2.0. The prototype of this model was implemented over FIPA-OS agent platform.

2.3.3 Agent Models versus Client Server Models

Most of the traditional applications are based on client-server model with a series of requests and responses between two parties. Agent model, on the other hand, projects a whole new way of solving problems. Instead of sending requests to the server, a representation of the client, that is its agent, moves itself to the server, executes its task and brings back the results. In this case all the client has to do is to create and send the agent to the destination and can proceed with its work until the agent comes back with the result.

In this way the Internet traffic is reduced, as there is just one bulk of code that moves to the destination. Moreover the agent model doesn't demand a continuous seamless connection with the client. The client may send the agent over the network and can get disconnected. When the client comes back after a period of the time the agent waits for its connection to deliver the result. When the agent carries cargo, which may be

a multimedia document, the size of the agent considerably increases. Under such circumstances the agent model requires a higher bandwidth. But nowadays since higher bandwidths are cheaper and more prevalent, the agent model proves to be the right replacement for the client-server architecture.

The agent, in addition to the above advantages, reduces the workload of the client and uses the resources of the server to complete its job. The agent model thus can be used in networks where the client has relatively lesser resources while the server is more powerful. The distributed nature of agents adds to its advantages of extensibility and reduced complexity.

In spite of all these advantages over the client-server architecture agents are under-used due to security concerns [10]. Security may probably be a concern for system designers while modeling an agent based system and moreover there are numerous bugs [8][9] that have been analyzed and reported but yet not being solved. There is also quite a considerable amount of research going on the field of agent security. Once the agents are proven safe and secure to be trusted, agent technology can set a new paradigm in ad hoc computing.

2.4 Ad-Hoc Communications and Mobile Agents

The basics of ad-hoc network were briefly discussed in the first chapter. This section discusses about ad-hoc networks referred to as MANET (Mobile Ad-hoc Network) and some of the issues that are encountered. The role of mobile agent in such a network and how context can be a part of ad-hoc network is analyzed.

2.4.1 MANET

A mobile ad hoc network (**MANET**) [55] is an autonomous system of mobile nodes connected together by wireless links. These mobile nodes move around the space and thus are prone to unpredictable changes in the network topology. The networks may also be connected to other stand-alone networks and to the Internet. One of the major concerns of such a network is the security of the nodes, as they are more prone to attacks like eavesdropping, spoofing, etc. The attacks largely arise due to dynamic re-configurability and the absence of a single centralized controller.

Some of the properties of MANETs are:

Autonomous: Each node in a MANET acts both as a host and a router. Since MANET is dynamically configurable, each of those nodes has to take care of the switching functions like that of a router. Since each node is autonomous the network becomes a distributed system. This in turn increases the robustness of the network considerably since there is no single central node on which the whole system depends upon. Hence even if there is a failure in one of the nodes the others can continue with their tasks with minimum or no disruption.

Routing: The nodes can follow single hop or multi-hop routing. In a single hop routing the target node is directly reached by the source in a single step whereas in a multi-hop routing the communication is achieved through intermediate nodes. Single hop is simple and doesn't involve complex algorithms unlike multi-hop. But each of these algorithms has their own merits and demerits depending upon the configuration of the network.

Bandwidth and Energy Constraints: Since the devices in an ad hoc network are mobile, they are restricted to limited use of power, which means that they may be available for a brief period of time. Moreover, the bandwidth of these devices is lesser when compared to static networks because they use wireless network.

2.4.2 Routing Protocols and Security

There are a number of protocols that are devised to address the requirements of MANETs. They are broadly classified [35] into two categories. They are table driven and source initiated on-demand driven protocols. Table-driven protocols are those in which the nodes are required to maintain a table to hold the routing information to all the other nodes. Some examples of table–driven protocols are DSDV [36] (Destination-sequenced Distance-Vector routing protocol) and WRP [37] (Wireless Routing Protocol). Another type of protocol initiates a route discovery process only when the source node initiates a request. Some protocols that fall under this category are AODV [38] (Ad hoc on demand distance vector routing) and DSR [39] (Dynamic source routing).

In DSDV, each node maintains a table with all the possible destination nodes along with the number of hops to that node. A unique sequence number denotes each of these entries to avoid loops. This routing table is updated periodically by transmitting the updates throughout the network.

AODV is an extension of DSDV, in which the periodical broadcasts on table updates are reduced. The source node, in this protocol, initiates a path discovery process when it does not find a valid path to the destination in its local table. It sends the RREQ

(Route Request) packet to its neighbors and it is further passed on to their neighbors until a new path to the destination is obtained by one of the nodes.

Security: Security in ad hoc communication is still at a nascent stage and the network is prone to a number of security challenges. The nodes in the network are susceptible to attacks both from inside and outside the network. The most common problem that the networks face is eavesdropping whereby a node passively obtains information from other nodes. The more serious problem arises when a node tries to corrupt or delete data. Another important concern in ad hoc networks is trust. Since the nodes are allowed to join dynamically to the network, every node must provide authentication to another before providing the service.

As proposed in [40], security comprises of *Availability* to protect the nodes from denial-of-service attacks, *Confidentiality* to ensure that the privacy information are not exposed to other nodes, *Integrity* of messages, authentication to prevent one node from operating under another node's identity.

2.4.3 Mobile Agents in Ad Hoc Communication

Mobile agents are capable of solving various requirements in an ad hoc network like routing [41], network management, and security. Since the network connectivity and communication are less reliable in an ad hoc network, mobile nodes can handover their tasks to an agent and wait for it to return with the results. Due to this capability of mobile agents, the network traffic is optimized as the agent carries out its tasks when the connection gets reliable and there is enough bandwidth to execute the task. Moreover,

mobile agent provides security at the higher level on top of the network layer reducing the security threats to the mobile nodes. Agents provide authentication of requests and maintains confidentiality of privacy information, which are deeply sought in an ad hoc network.

Since the mobile nodes mostly run on batteries, power saving becomes another important criteria. The agents play an important role in reducing the power usage by carrying out the tasks on behalf of the node even after the mobile node has left the network. Thus both from the user point of view and the network point of view, mobile agents can be employed for various operations in the ad hoc network.

2.4.4 Context in Ad Hoc Environments

Context plays an important role in managing ad hoc environments. Since the whole environment is dynamic and re-configurable, the knowledge of the related entities in the ad hoc environment becomes essential. Capturing the context, analyzing and matching context data from various related resources enable the collaboration of various devices. Context can address the security issues in an ad hoc space by acting as a firewall against entities possessing irrelevant or questionable context. The model that will be discussed can capture the context in the ad hoc environment and facilitate collaboration of related entities spontaneously or through explicit requests.

2.5 Context-aware Projects

This section details some of the existing agent-based context-aware models and show how agent technology can be profoundly used in context-aware computing. The

projects that shall be discussed are *MyCampus* [11] project from Carnegie Mellon University and *Genie of the Net* [12] from VTT Electronics and the Intelligent Systems research group of the University of Oulu.

Both these projects are aimed at creating a context-aware environment by using agents. The former discusses context-aware agents discovering and presenting internet and intranet services to the user while the latter is about context-aware information management and presenting it to the user.

2.5.1 MYCAMPUS

The project is currently being developed at MIT focusing on mobile agents that discover internet and intranet services and provide them to the users. These agents carry out their tasks taking into consideration the user's preferences and context attributes. The context attributes are the user's location, date and time, etc. The users can control the accessing of their personal resources with other users through permission profiles. Personal resources are the values of context attributes, which other users may request for specifically or subscribe for notification of changes. Tasks are represented as goals and relevant operations are triggered when a new web service is discovered. If the web service that is required to perform the task is not found at the current location, the agent instantiates another service discovery process to search for the service in the nearby locations on the basis of current context of the user or task.

This research group has implemented a simple agent that can guide the user to the restaurant based on the user's food preferences and the appointments that the user might have during that time. The agent, for example, suggests the best place to have lunch

taking into account the location of the user and the restaurant, the choice of food, the whether, and the appointments that the user might have on the day.

2.5.2 Genie of the Net

This project is primarily focused on context-aware information management. It is an agent-based architecture that deals with management of services on behalf of the user in an *intelligent environment.* The intelligent environment according to the authors is an environment studded with sensors, actuators, devices and services.

The architecture is based on four agents. The Sensor agent captures the sensor data and the context agent derives the user's context from them. The active user agent converts this data into information and makes them available to the users through the user interface agent. The context information is stored in *context variables* and a set of these variables with their relationships form the context history in this architecture.

The application area that this group is working on based on this architecture is the management of calendars and notice boards. *Notice board* is a collection of information presented to the user in a specified context and *Calendar* orders the events on basis of time. One of the applications of this architecture is the health club. The health club application is designed to assist the user in his cycling exercise. The user is assumed to have designed a schedule for each exercise. The exercise schedule is presented to the user automatically in the specified context and reminds about the forthcoming exercise. During the exercise the context data such as heartbeat rate and bike's speed are recorded and stored into context history. The system analyzes the exercise and presents the collected data, the context history, and the results of the analysis to the user.

2.6 Related Work

A very similar approach of managing context in mobile ad hoc environment discussed in [19] is about the design of a middle ware to support context awareness. The middle layer provides APIs to application layers for them to obtain discovered context. This architecture mainly focuses on discovery and efficient delivery mechanisms of context data rather than processing them. The sensors considered in the design are physical sensors that obtain physical environmental attributes like presence, temperature, pressure etc and do not handle context history. In CMS, agents perform context delivery and context is handled in more detail where derivations are performed from context history. In addition CMS also has software sensors to monitor network activity and detect new software services.

Castro et al [46] discuss an infrastructure for smart spaces containing sensors. The infrastructure consists of three main components, the inference server for calculating probability distributions, a lookup service for keeping track of different sensors and services, and a memory element for recording the results of sensors. They use sensors to provide fusion service, a service, which delivers appropriate context for various types of applications making them context-aware. The context information in this infrastructure is mainly presence and location of mobile users whereas CMS performs location discovery and delivery of services to the requester. Fusion services are specified as Bayesian networks in XML documents. When an application needs a fusion service it checks the lookup service and when it is found, the application downloads an interface to the service manager which delivers the service based on cost constraints. The CMS that will be

discussed is agent based and does not restrict to the presence information of the user but takes a step forward in identifying and deriving entity attributes.

Another context-aware model for ad hoc networks that also uses agents is described in [47]. It follows a similar approach of creating views for every agent. This view will contain all the relevant context data for a particular situation and is constantly updated by the underlying middle-ware. It is left transparent to the application layer so that the programmer can set the view size and scope.

Each individual agent is allowed to create its own views with context data that is more relevant to its functional behavior. The context data is obtained as a reference object where the agent finds the context object by requesting the bindings to object in views. The agent can use this reference as long as it is alive and once the object expires, the agent has to make a fresh request for that object. There is a more flexible representation of context with a custom specific data structure and how agents manipulate the views to satisfy its requirements. In CMS the agents themselves form and exchange what is called as context object for every entity. The agents terminate the associated objects once the entity has left the room environment. The agents are strongly bonded to the entities in CMS.

In all these models including the CMS, the burden of updating context is shifted to the middle layer so that the application layers can just focus on the data rather than its maintenance. This sort of approach brings about an organized context management thereby improving the performance of the model.

2.7 Summary

This chapter provided the required knowledge to apprehend the contribution of this thesis. The tools and techniques needed for capturing and representation of context were discussed. The chapter also explained some relevant techniques like service discovery and agent technology. The likes of mobile agents in ad hoc networks and how agents tune into the various operations performed in an ad hoc network were discussed along with how context could strengthen the usability of these techniques. Finally a couple of projects that emphasize the use of various techniques to create context awareness in ad hoc environments were mentioned.

Chapter 3

Context Detection and Management

The two very broad categories of tasks in context-aware computing are capturing or acquiring context and managing the context data. The next step is to explore these two categories and understand the rational behind managing context data. The chapter is intended to introduce the various tasks and requirements of context manager. Context detection and management act in concert as management is based on the collected context data type and the intelligence behind collecting context is provided by context management. The chapter also explains various tools and techniques that are used during the various phases of detection and management process along with some examples of similar existing models. This chapter is mainly intended to provide an understanding about acquiring and processing context and the responsibilities shouldered by a context management model.

3.1 Introduction

The primary step in any context-aware application is to identify the set of context attributes that are relevant to the situation, or in other words, would describe the particular situation. The situation here may be characterized by time, location, by one of the participating entities or any other factor affecting the system. The following sections deal with the mechanics and tools that could aid in capturing and representing context information in such a way that it facilitates management.

Management basically is *deriving meaningful information from raw context, storing and maintaining that information to provide them to the applications that request the same.*

3.2 Context Detection

Context detection can be broken down into two sequential steps namely Context defining and context acquisition. The context parameters that need to be captured are first analyzed and then the specific sets of parameters, which are substantial, are captured. The sources of obtaining the context are different types of sensors.

3.2.1 Defining Context Parameters

The first and foremost step that is performed in designing a context-aware application would be to identify the various context parameters that provide a required level of description in a particular situation. This comprehensive list of context parameters can be enumerated by carefully analyzing the requirements.

A simple example is a word document described by author name, date of creation, number of characters, and number of times edited, etc.

The list can be exhaustive when the application gets complex where one context parameter can hold multiple values for a same context attribute depending on the situation. Thus there has to be a specific subset of context parameters that are substantial to describe a situation, which are grouped as *context views* [43]. Context views are similar to views in database, where we select particular fields or parameters to describe the situation.

The main challenge lies in identifying the context view from the galore of possible context parameters. It arises mainly because of the spontaneity and dynamic nature of the environment, where these context views should be decided on the fly. The most reliable and simpler way to answer this is to maintain larger views and thus reducing the number of combination sets of context parameters. By having a larger view, lesser number of such views can be pre-defined which can suffice unexpected system states and requirements. The disadvantage of this method is the processing time of the views. If the view is large, the number of context attributes that should be captured and processed increases and thus increasing the processing time.

The context defining phase, therefore, forms a crucial part in deciding the performance of the whole system and hence care should be taken while defining the context parameter set.

3.2.2 Acquisition Techniques

Once the context data set is defined, the next step would be to find ways of capturing the data. This section explores the two broad categories of context capturing techniques namely Hard Sensing and Soft Sensing. The applications may use one or both of these techniques depending upon the context to be captured.

3.2.2.1 Hard Sensing

The context information, like the presence and the physical behavior of the entities in the environment, are captured by hard sensing techniques. Some of the tools that fall under this category are cameras (Image), RF/IR Sensors, GPS and many more. Simple applications use image cameras to capture the presence of a person and more complex ones detect the orientation of the person and emotions like smile or a nod. Hard sensing generally provides data at its raw form and requires processing on them to get a meaningful description.

3.2.2.2 Soft Sensing

Unlike hard sensing, soft sensing refers to methods of obtaining context through software or programs. Some examples are getting the system and login information, capturing bandwidth and network connectivity, getting users preferences etc. Soft sensing may be performed at various levels of the context-aware system since input to these types of sensors may be from a variety of sources. For example the identity of a user may be obtained from the user's login information or from the user's preferences depending upon the situation.

3.3 Basic Requirements of Context Manager

The requirements of the context manager with respect to an ad hoc user collaborative application are enumerated, which is the primary goal of this thesis.

- The context manager is responsible for obtaining the context information from relevant sensors. The context manager should be compatible with the variety of sensors and be able to gather the raw context from them without loss of data.
- Must be able to obtain meaningful information from raw context. Moreover, the context manager is responsible for maintaining and manipulating context data. That is, context manager must act like a container for context data performing storage and updates.
- Context manager should be able to trigger appropriate actions resulting from a context change. When there is a specific pattern of context information that demands an action, context manager must be able to identify this and act accordingly.
- Context data security is another responsibility imposed on context manager. It has to act as a shield for context data from unauthorized requests. All the communications relating to context data exchanges must be monitored so that the privacy of any entity is not exposed.
- Context manager acts as an interface to upper layers of application. It provides context information for other third party applications requesting them. The request for this data can be in any format ranging from a database query to an agent request. The context manager should be equipped with ways of interpreting the request and providing result understood by the requester.

3.4 Tasks Performed By Context Manager

Having seen the various requirements of the context manager, the next step is to discuss the tasks performed by context manager to address those requirements.

3.4.1 Context Representation

Context data once obtained from the sensors should be represented in a convenient form so that it provides an unambiguous and convenient retrieval of data. The two questions to be answered by context representation are "what to represent?" and "how to represent?" The former question has an obvious answer, i.e. the object or entity of interest. How the entities are going to be represented in the question of concern. Representation can be thought of as a two-step process – Grouping and Representation. The representation of context about a particular entity should be based on some criteria. That is, clusters of context data that are related in some way must be formed for representation. In our case, we can group context entity based on time and location and represent it by key-value pairs.

3.4.1.1 Location-based Representation

In a location-based representation, a set of context values about a particular entity relevant to the location are grouped together. This attribute set does not change for that entity unless it leaves that location, but the values for those attributes change appropriately. This is a simple grouping that has a larger context view and does not encounter constant attribute changes. Thus in a location-based representation, the context manager has to listen to only a specific set of sensors constantly.

3.4.1.2 Time-based Representation

In time-based representation, the context attributes are grouped with respect to time, that is, grouping a particular set of values associated with user at every instance of time. This means that the attributes that are associated with an entity at one point of time may not be there at the next instance of time. The attribute set is constantly updated at specific time intervals. In this representation, the context view is relatively small compared to location-based, but the context manager undergoes a lot of switching between sensors due to the constant changes in context attributes.

3.4.1.3 Key-Value Pairs Representation

Key-Value pair is the most convenient way of representing context information. The key is the context attribute of the entity and the value can be single or sets of possible values for that attribute. All the context information is stored in the form of attributes and values. Such a set of attributes and values of a particular entity can be clubbed together to form an object or a file. This object or file is made accessible to the various modules in the context management system.

3.4.2 Semantic Formation

Semantic formation is obtaining meaning or providing understanding from the captured context. The input obtained from the sensors does not provide any meaning as they are discrete values. The context manager has to compare and coordinate the context data from various sensors and resolve them into formats understood by the other modules that act upon the context information. This can be considered as a crucial task of the

context manager since the result of this action can affect the behavior of the system. There are plenty of chances of ambiguities during semantic formation when performing comparisons with multiple context data.

Semantic formation is primarily achieved by forming relationship among context values. Each context data is associated to a subset of context attributes by relevance. The context manager looks for such relevant context from the context view of each entity. The values of one or more of the context attributes considered may contradict while deriving semantics. Under such circumstances, the context manager should try to compromise for contradicting attributes by prioritizing attributes based on the situation.

3.4.2.1 Triggering Actions

The context manager is also responsible for executing tasks based on derived semantics. Once the various modules in the context manager has come to a unanimous agreement in describing a situation, the context manager checks if there is any task demanded by this situation. The task may be triggering an action or just passing the information to appropriate module.

The following example scenario best explains this task. Consider a user whose context view consists of task queue, time, and preference. At a particular instant the time matches with one of the tasks in the queue, which, as in this example is sending a fax. The context manager recognizes the task and checks with the policies of fax service that is ready to use, making sure that the user is authorized to use the service and finally triggers the service to send the fax.

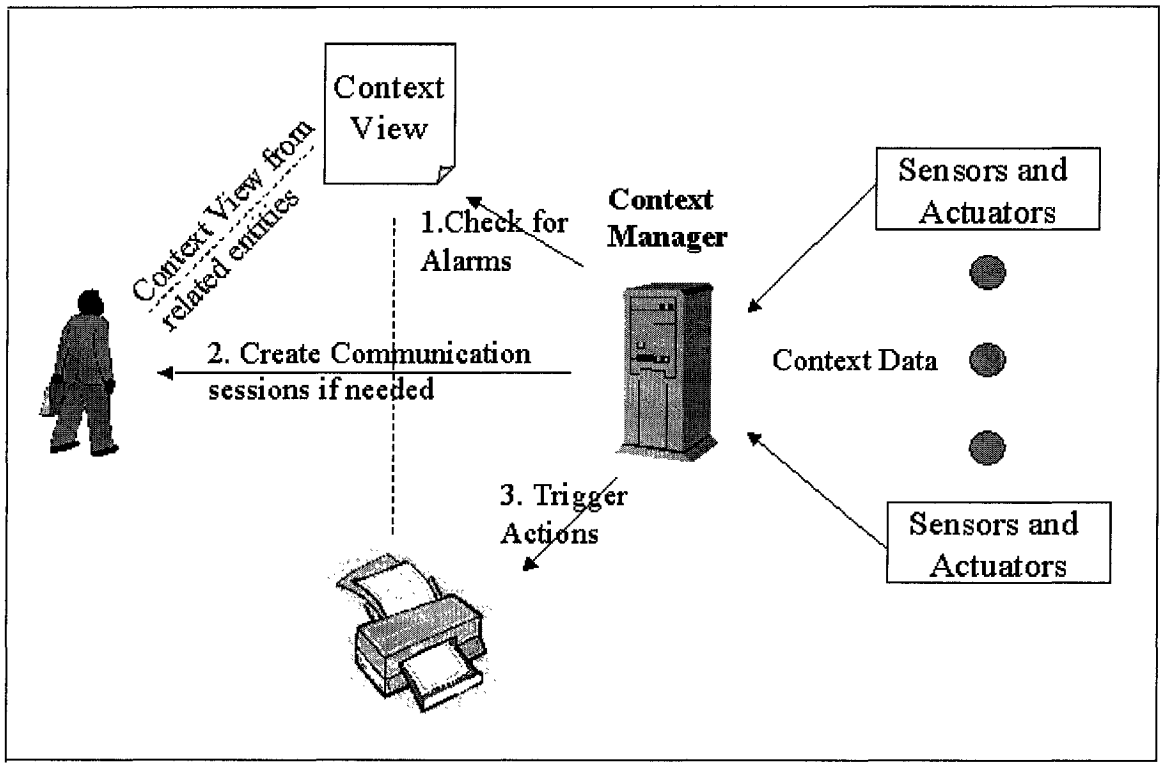

Figure 3.1: Context Manager Triggering an action

3.4.3 Maintain Context History

Context history forms an important part of information base in the context management system. It archives the activities and behavior of entities in the environment. The activities archived may depend upon the scope of context in the environment. The more the context is involved in controlling the entities and their behavior, the more detail the entity description gets.

For example, if the context is just responsible for the availability of the service, then the other entities that use the service will not be within the context views and will not be archived. On the other hand if the context is responsible for controlling and

managing the service, then the users involved with the service, the service status etc. will appear in the context view and be archived.

More than just a mere storage, the context history acts a learning source to understand the behavior of the entities. Archiving paves way for achieving *smart context*, where the context information is manipulated or predicted from the gathered knowledge. Smart context helps not just in automation of services but also in adding thought and sense to the automation. The context manager will be responsible for deriving this context to carry out its tasks through the primary use of context archive.

3.5 Context Management Models

So far the various aspects and requirements of context manager were discussed and now the focus shifts towards some of the ways in which context management has been designed. Indulska, J., et al [43] presents a very similar approach of a context management system that supports pervasive computing based upon the CC/PP standard. The context manager acquires information from awareness modules and interprets the context so that it can be used by context aware applications. The mandatory functionalities of the context manager that they propose are -

- Acquiring context from various sensors
- Creating context information from the acquired data
- Manage the context information in a common format.

The context manager also maintains a persistent storage of context to provide the third functionality.

Another Context management mechanism [45] is focused on solving problems in local and distributed transaction processing among various applications. The context refers to the grouping of resources in a system to solve a task or provide interrelationship among resources. The context management model is fully involved with soft context, which is related to a system and its resources like application threads. The context manager helps applications manage their local or distributed threads that solve their task.

3.6 Summary

Context management constitutes the central part of a context aware system. The basic requirements of a context manager and the various tasks that the context manager performs were detailed. Context management can be understood by different meanings depending upon the application. The tools and techniques of executing the task may vary among context managers, but the objective remains the same. Context managers may include optional modules like context archive or a persistent storage as this just adds to the feature set of the context manager. However some applications may require them to be a mandatory part as it may influence the behavior of the application considerably. Finally some of the implementations of context manager, their requirements and objectives were discussed. This prepares strong background knowledge of the architecture that will be presented in the ensuing chapters.

Chapter 4

Context Management System Design

The chapter presents our work in designing a context management system for ad hoc user collaborative applications. This chapter holds the central theme that represents the major contribution. This chapter explains an architectural model that is targeted towards providing various features of a context management system. The rest of the chapter is organized as follows. First the goals of context manager are defined followed by an overview of its features. Then the various modules involved in the context manager are presented with the different representations of the architecture. Finally the issues that arose from this design are analyzed.

4.1 System Goals

The Context Management System (CMS) is designed to support applications that are ad hoc in nature and are targeted towards aiding collaboration between users and services. The CMS has to detect and process context data from various sensors and provide persistent storage of context. The context manager must '*explain*' the situation to all the associated modules that interact with it.

A situation can be thought of as,

"A particular state in an environment which is best described by the properties of a single or group of entities bearing relationship with one another through common context attributes."

By '*explain*', it means that the CMS should be able to procure all the data related to the requester and the situation and dispatch them on time. In addition to data delivery, the CMS should also be able to trigger actions if demanded by the situation. Last, but not the least, the CMS should not compromise with the ad hoc nature of the environment while making it context-aware.

4.2 Overview of CMS design

The context management framework helps in acquiring context from variety of sensors and building information base by relating associated context. This section presents an overview of the architecture of context management system comprising of various blocks that performs specific context management tasks. According to CMS architecture, the context detection is an integral part of context management system.

The following represent a general set of steps that are followed while designing a context management system to solve the above goals.

- The first step is to identify the entities that are involved in the application and those entities whose change-of-state has an effect on the application
- Then one has to find ways of capturing those changes with the entities. The capture may be soft capture or hard capture.
- When the capture is done, the efficient ways of representing the captured values has to be looked upon. This also affects the storage of this representation, for example, File storage or database tuples.
- Relationships have to be drawn among the entities so that a change of state of one entity is identified by the related entities.

Once the changes are identified then the application has to trigger corresponding actions

4.2.1 Context Management Framework

The context management framework is targeted towards information gathering and management. The most crucial part would be to identify the various kinds of information that needs to be gathered. Some of the key information required are,

- **Presence of the entity**: This indicates if the entity is present in the current physical location or not.
- **Location**: Description of the physical location where the entity is being spotted.
- **Date and time of presence**: This holds the duration of presence and the date the entity was spotted in a particular room

- **Basic profile**: The profile of the entity consists of a basic set of variables that are initially required for unique identification of that entity.
- **Preferences**: Preferences contains sets of values for a particular variable whose ordering is based on priority. For example entity A has an attribute A1 where,

  A1 = V

  V = { $y_1, y_2, \ldots\ldots, y_n$ } where y_1, y_2 = Values of attribute A1 with priorities 1,2 respectively.
- **Relationships with other service/people**: This defines the relationship of an entity with other entities. For example the owner of Service S1 is User U1. It means that the value of attribute owner is a link or pointer to another entity, which is a user in this example.

The following paragraphs present an overview of Context management framework and discuss in lesser details the various modules that combine together to form the CMS.

The Context management framework consists of 6 major blocks.

1. Sensor Block
2. Data Storage Block
3. Query Handler
4. Agent Deployment Block
5. Filter
6. Application Interface

The *Sensor Block* abstracts the physical sensors from other modules. This block is responsible for converting physical context data to specific formats understood by requesting modules. The sensor block utilizes the local database for generating meaningful context.

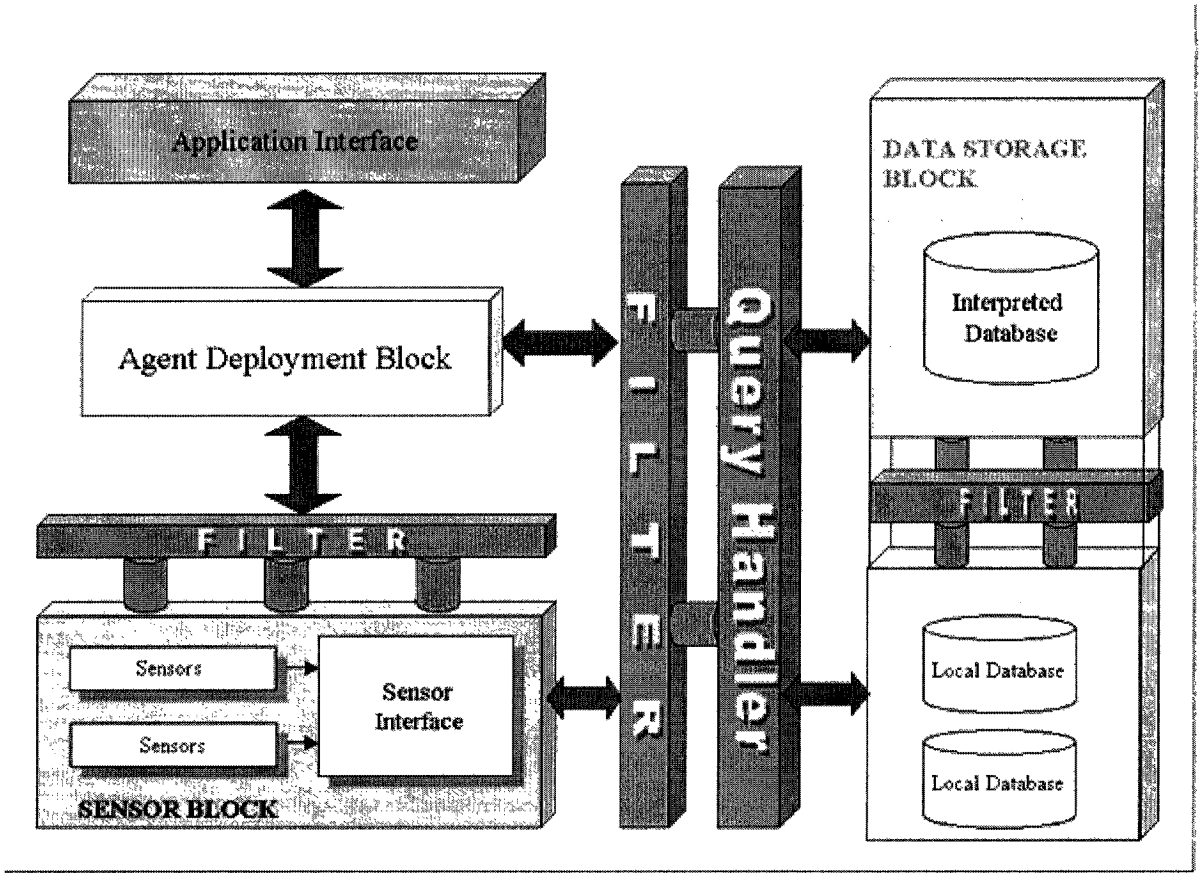

Figure 4.1: Context Management System – Block diagram showing internal CMS modules

The *Data storage Block* (DSB) acts as the information base containing context data stored in various structures. It is the responsibility of this block to manage stored context and maintain the integrity of data. Efficiency in memory usage while reducing data redundancy is another important aspect to be considered by data storage block.

Query Handler (QH) is closely associated with data storage block since it handles the request and replies to the database. The request and reply format may vary depending

upon the requester. The requester may be one of the CMS blocks or external agents. The query handler should resolve the request type and provide a suitable formatted reply.

Agent Deployment Block (ADB) is the core of all the operating agents. New agents are spawned here and it also acts as the gateway for other external agents. The agent deployment block controls all operations performed by the agents. The applications that reside on top of CMS communicate with the internal blocks through ADB.

Information exchanged between all the blocks need not always be genuine and meaningful. There may be unwanted or erroneous information passage taking place between blocks. To check this problem, a *Filter block* is introduced, which filters such unsolicited data from reaching the blocks. Filter also performs security checks on requests and replies through policy management. The role of filter block changes depending upon the block this is plugged into. For example, the role of filter plugged over sensor block is not the same when plugged over Data Storage Block. The filter over sensor block avoids reporting of unnecessary redundant context change, while in latter it maintains integrity in context data stored in local database and interpreted database.

Application Interface block deals with communication with external application that are agent based or non-agent based. It also provides APIs for generic applications to interact with the CMS.

4.2.2 Feature Sets

The following sections discuss some of the predominant features of the CMS Architecture. These features provide better understanding of the CMS, its goals and capabilities.

4.2.2.1 Distributed Design

The context management system comprises of modules at various terminals or geographical locations. This creates a model where all the blocks work autonomously at one or more locations. For example the sensor and data storage block need not necessarily be on the same system or in a same location. The agent model automatically achieves this distributed design. Each of these models operates independently and collaborates with one another through agents and hence the efficiency and reliability of the system is improved. Each module can have its own backup system to provide fault tolerance.

The whole CMS as a system is also a distributed structure. This means that entities present in two different ad hoc networks in different geographical locations can be projected to be in the same room or network and can function together through a common CMS. Ideally, a service that is available in a remote network can be provided access to other entities in different networks that possess accessing rights.

4.2.2.2 Context Representation

Context is represented in variety of ways across different modules. This is because every module has its own set of tasks, which may demand a specific representation for better context data handling. The ADB may represent context as a hashtable object while the Data storage block may store it as a database tuple. Each module shall be equipped with their own conversion mechanism to identify and convert data into preferred formats.

This difference in context representation arises due to the fact that each module executes independently and does not have to depend on the mechanics of operation of related modules as long as the modules understand a common language for data communication. Another reason for this difference is the variety of forms of context information ranging from simple key-value paired profile to complex policy representations.

The description of each entity can be broadly categorized into profile, preferences and presence. Profile of each entity contains the properties of the entity that shall describe

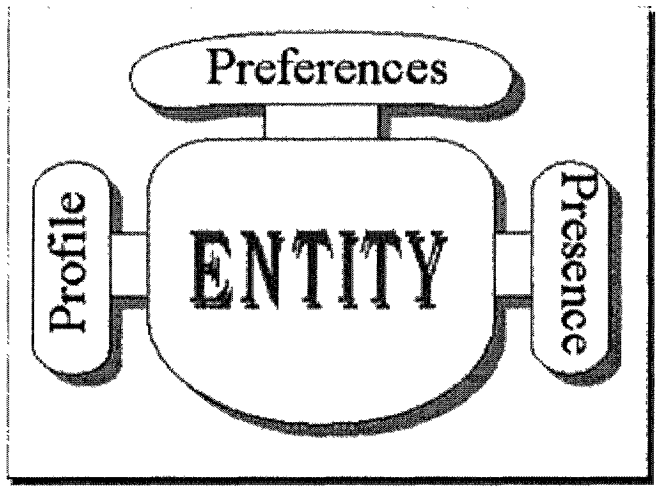

Figure 4.2: Entity Description

its physical nature. The three common entities in our system are people, services and the location. Each one of them may or may not have sub categories. A service profile contains some of the basic information such as ServiceName, Owner of this service, location of this service and advanced information on the properties of the service, its capabilities and configurations. Figure 4.3 shows a sample service profile representing a printer.

The user profile similarly contains basic information such as Name, address, Home number, Designation, location etc. and some more parameters such as the status of the user, the services belonging to this user and security parameters of this user.

```
<xml version="1.0" >
 <Document>
<Type value="Service" />
<Device>
      <Device_group value="PRINTERS" />
      <Device_name value="HP LaserJet 4050 N PS" />
 </Device>
<Properties>
      <Color value="no" />
      <sided value="yes" />
      <Memory>
            <minMemory value="8" />
            <maxMemory value="11" />
      </Memory>
      <Resolution value="1200" />
      <PS value="true" />
 </Properties>
 <Owner value="CBY-B502" />
 <EnteringTime value="" />
 <ExitTime value="" />
 </Document>
```

Figure 4.3: Printer Profile

By having a closer look at the profile, it can be seen that all the information can be further classified into two divisions, namely static and dynamic parameters. The static parameters are the ones like the name, address etc. which remain constant. The dynamic

ones are those like location, status of the user (available, busy, invisible), owned services etc. The dynamic properties are of most interest as they should be constantly updated and propagated to the upper application layers. Most of these properties can be changed manually by the user or automatically depending on the policy settings of the user.

The user defines a set of policies whenever he/she is assigned a tag. The policy settings control the users preferences and help change the user profile dynamically. The preferences list out the priorities or bias that each entity may hold to other entities.

As a simple example, the user might say

On(*Day = Saturday)*

If *(Location = Room1) and*
(time between 12:30 and 14:30)
Then *user.setStatus('invisible')"*

A set of such policy strings can be applied over the user profiles to make the user profile more smart and adaptive to the environment. The unavoidable task here is that the user should manually enter the profile and policy of the entity.

4.2.2.3 Agent Model

All communications between internal and external modules are purely agent-based. The agent model provides an abstraction to the underlying layers and avoids communication complexity. Abstracting communication complexity means that more than just messages the data object exchange mechanisms are also hidden. In other words each agent acts as a *wrapper* to every entity.

There are two types of wrapper agents in this system. Those types of agents that wrap people are called *Presence agents* and those that wrap services are called *Service agents*. These are the agents that interface with the user and communicate with other agents. These agents also provide security features by authenticating every request for these entities. The only way to access the entities is through the agents. The agents respond only to calls from those agents that are legally registered with this environment.

Whenever there is any request for a service, the service agent loads the tools required for completing a job. For example, when there is a request for a printer the service agent associated with that printer fetches the file to be printed and uses the printer driver for getting the job done. The accessibility of device drivers is not handled in detail as it deviates from the intended target.

The main purpose of presence agent is to manage the preferences of the user and create awareness about the user among other agents involved in the collaboration. The presence agent holds the contact information of the user and monitors his activities. This is not a personal agent and does not carry out the activities of the user. But if there is already a personal agent for that user, the presence agent delegates the responsibilities to the personal agent.

Figure 4.4 illustrates the usability of agents in this architecture. The profile of the person or service contains all the relevant information required to identify the service/person uniquely. A sample profile that describes a printer shown in Figure 4.3 is one good example of description of a service. The agent maintains the entity attribute information in the profile to distinctly project itself as the representative of the entity. Moreover the policies that state the rule the entity is supposed to follow and its

preferences are also encapsulated by the agent since the agent is the one that will take over the actions on behalf of the entity. Thus agents can be considered as wrappers that are formed from the available information regarding the entities. The Agents are created and dispatched when the presence of an entity is sensed. The agent remains active as long as the entity it represents is active in the environment.

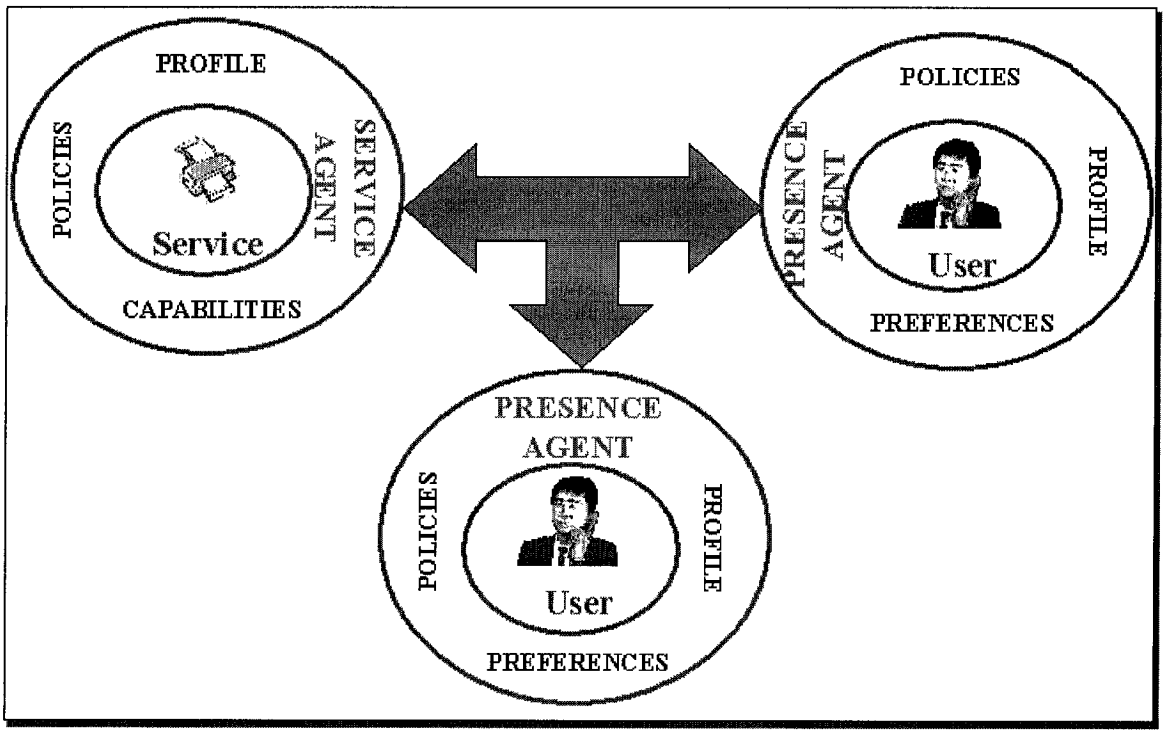

Figure 4.4: Role of agents in CMS

4.2.2.4 Context History

Context history can be considered as an important feature of the CMS. Every action performed by the entity is stored in the database along with the timestamp. That is, all the activities performed by the user between entering a room to leaving the room are recorded. The context information of each user is also archived for a particular length of

time depending upon organizational policies. The archive forms the main input for creating a context-aware environment by displaying patterns that assist in understanding the behavior of an entity.

For example the status of the user is set depending upon the previous status values of the same user.

The distribution of the user status for a particular length of the time is drawn and the probability of each status at that point of time is noted. The status of the user is automatically assigned by a random selection. This random selection is achieved by applying the probability percentage which is directly proportional to the number of occurrences of that status at that particular point of time.

Mathematically,

$$Pit = \frac{C_{it}}{\sum_{i=0}^{n} C_{it}} \times 100$$

P_{it} is Percentage of occurrence of status i at time t.

C_{it} is Count of occurrence of status i at time t.

n is the total number of states

With this probability, the status *i* is applied at time t. The user can change the state back to his/her preference. The same formula applies not only for status but various other attributes. The count of the number of occurrences of a particular attribute (C_{it} in the above case), may involve one or more conditions. For example,

C_{it} may also be

Count of number occurrences of status i at time t ***and***

Room_name is "Room1" ***and***

Services_possessed = false

The presence or the service agent makes up this conditional statement whenever a change in context information is detected. Each service or presence agent takes exclusive control over the archive of that particular entity. Another important fact here is that

"If there is a conflict between the user-defined policies and the derived value, the policies have a higher priority and the derived value will be ignored".

Users also have the capability to query the archive of other users. The level of query is restricted to the privileges allocated to every user. It is denoted by the query level, which varies from person to person depending upon the designation and ownership attributes of users. These values are defined during the creation of the profile and are subject to change by privileged users.

4.2.2.5 Multiple Sensor Support

The CMS architecture can be enhanced by incorporating multiple types of sensors without significant design modification overhead. The sensor block supports diverse hardware and software sensor plug-ins even during runtime. They may not be exclusive sensors, but may also be other applications that provide relevant context data as a part of their function. The CMS identifies them and creates communication sessions to acquire data from those programs.

4.3 Detailed System Architecture

The overview of the architecture provided a brief explanation on the various blocks in the CMS and did not get into the working mechanisms. The following sub sections provide detailed system architecture with thorough explanations on the various participating modules.

4.3.1 Sensor Block

The sensor block forms the main source of context information input to the CMS. The sensor interface forms the central part of the sensor block involved in coordinating with various sensor types.

The sensor platform mediates between the physical sensor data and application oriented data. The sensor platform comprises of separate sensor controllers for hard sensors and soft sensors. The sensor controllers basically monitor and control the sensor attributes and activities.

The hard sensors generally accompany with them their sensor controllers, which can be plugged into the sensor controller block to control that specific sensor. These types of external controllers may require additional module that identifies the data stream provided by those sensors. In addition to sensor control, the sensor platform also has the responsibility of answering to queries from other CMS modules relating to sensor information.

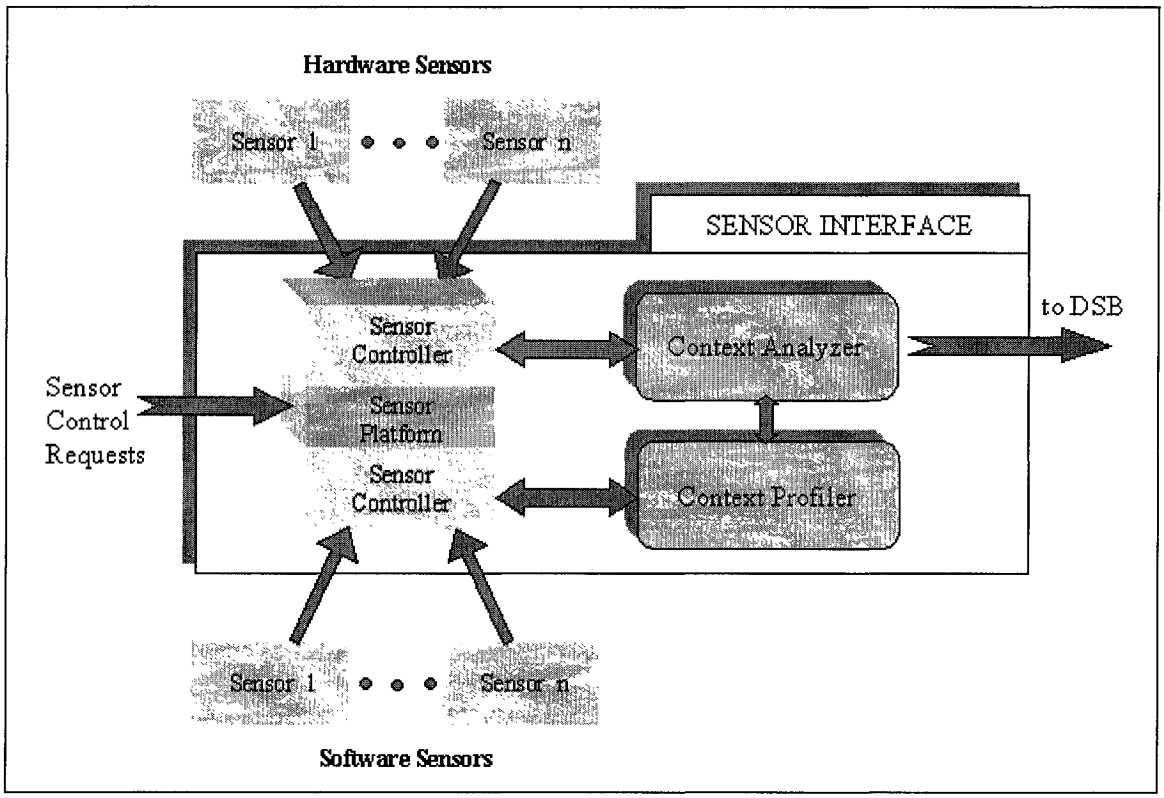

Figure 4.5: Context Sensor

The context profiler holds the basic predefined data about the sensor characteristics and sensor data type. The sensor controller may also populate this profiler with information specific to any sensor. The context profiler is designed for usage by the context analyzer and the controller modules regarding the sensor profiles. The format of the sensor profile could be a simple XML file that indicates the ways of connecting to the sensor controller to obtain data stream.

The context analyzer holds responsibility for communicating with external sensor controllers to convert their data streams to application-specific data in the sensor profile format. The context analyzer obtains sensor specific information from the profiler like sensor type, data stream type, socket information etc.

For example, in the case of an image sensor, the sensor analyzer may look for the data port and the image data type of the image sensor controller.

The context analyzer is preloaded with various data acquisitions methodologies that span the most common sensor types so that the analyzer will be aware of the necessary attributes in the sensor profile. The analyzer converts the data obtained from the controller to suitable format suggested by the data storage block and forwards it to the data storage block.

4.3.2 Data Storage Block

The main task of the data storage block is to create and manage data in its original and converted formats. The DSB uses query handler to convert data between context objects, database tuples or Agent Communication Language messages.

The local database contains context attributes of entities that are used to uniquely identify the context change. This could be a relational database or a file management system consisting of profiles of various entities. The object database contains context objects describing each entity in the environment. This is a custom database designed to store and retrieve context objects through APIs or procedure calls by external programs

Context Objects

Each entity is described as a set of attribute-value pair in a context object. Some of those attributes may further be described by sub pairs. These values in the pair may be pointers to other context objects. A binary tree representation of the context object is followed since context object follows a XML file structure.

Moreover, since context objects are prone to frequent accessing, a tree representation will prove to be an efficient data structure to describe this model. Agents create the context objects. An agent may create one or more objects to describe an entity. Each context object has a pointer to its creator and is available for other agents or applications trying to access this entity.

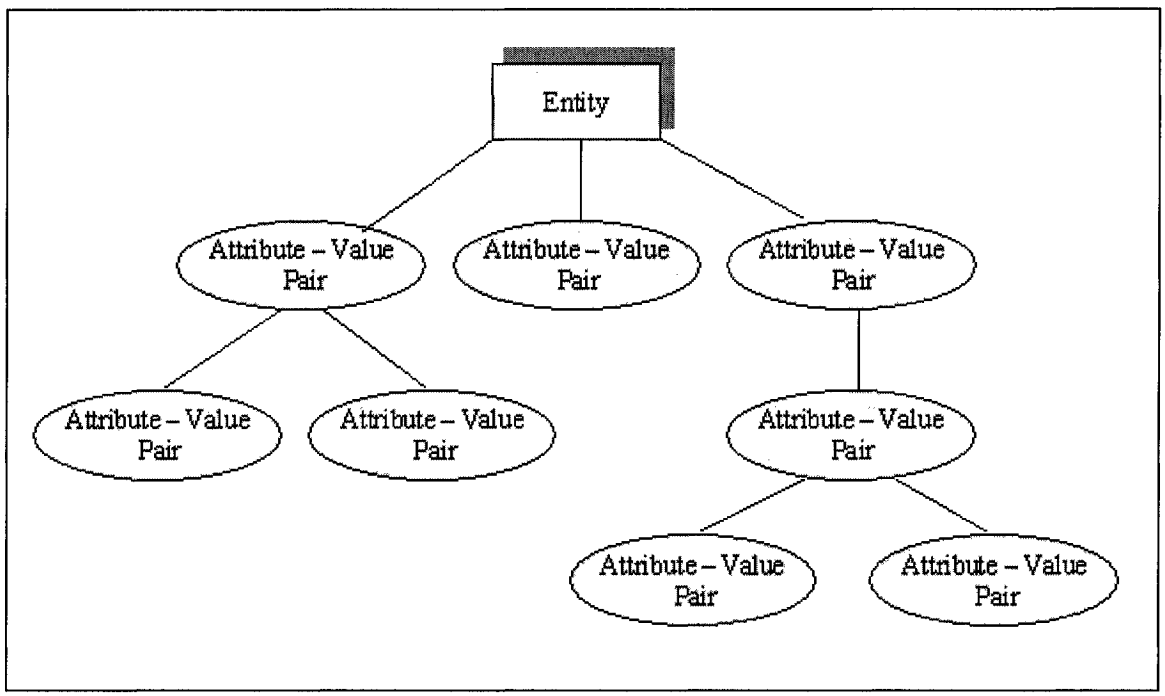

Figure 4.6: Representation of Context Object

Context objects may be cloned or duplicated whenever required. The changes made to the clone shall be reflected in the original object whenever the entity manipulating the object demands for it, i.e. the clone may or may not be synchronized with its parent. A context object shall be terminated by its parent agent or by itself. The reasons for this would be

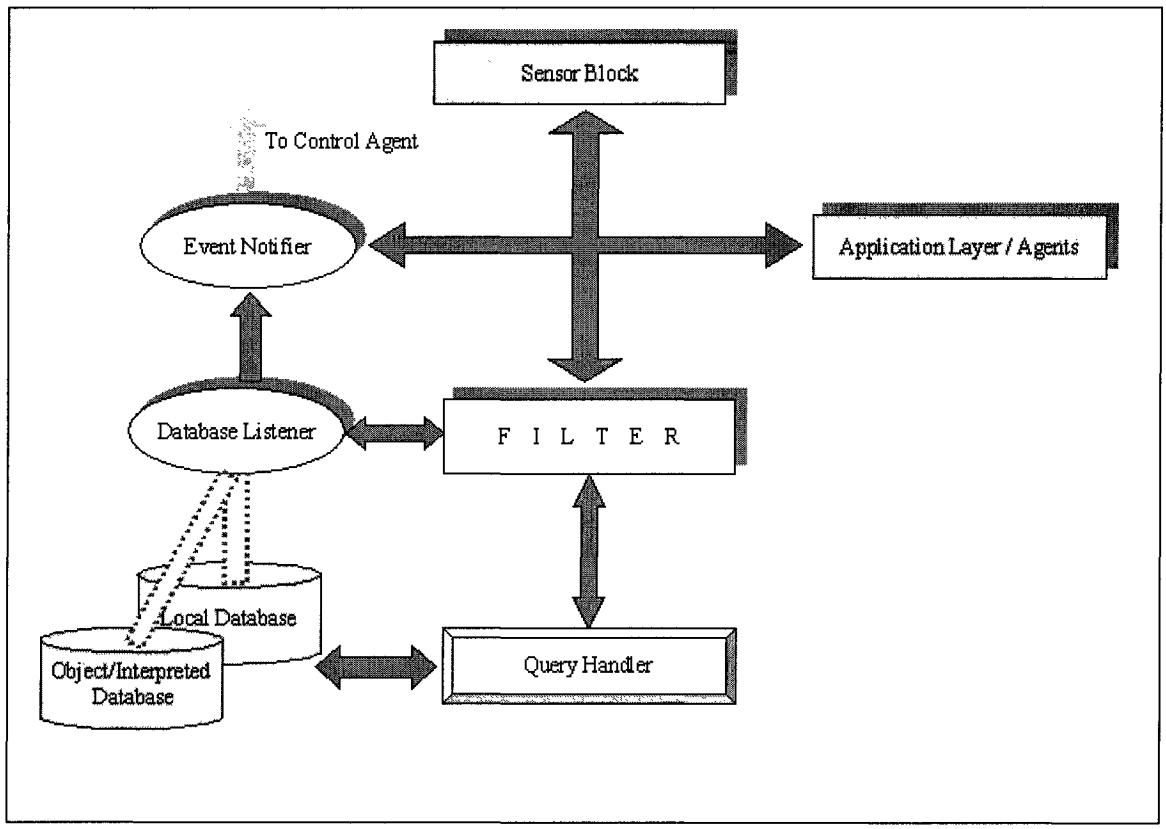

Figure 4.7: Data Storage Block

- Expiry of time-to-live associated with every object
- Absence of entity in the environment
- Termination of parent object if any

Before termination, the context object transforms its latest attribute values to local database profiles. The same profile is overwritten each and every time the object extinguishes.

Listener and Event notifier

The task of the listener attached to the database is to identify the changes in the attribute values in the databases. This listener is a constantly running thread program that

checks for the attribute values of entities. The database listener is comprised of one or more such listeners. The agents representing the entities may themselves act as listeners for the attributes of host entity. Therefore the database listener may be visualized as a pool of listeners each of them monitoring their own entities. The listener agents register with the database listener to gain access to the entity attributes. This registration automatically expires when the entity leaves the environment. The event notifier is designed to inform or carry out the corresponding actions arising from the context change. The event notifier is itself an agent that monitors the attributes and preferences of specific entities. It acts as a broker between two or more agents in a situation that demands collaboration of multiple entities.

The agents responsible for individual entities are focused upon the activities of that particular entity and are unaware of the status of other agents unless informed. The event notifier performs this job of creating awareness among the agents about their related agents.

The agents or programs in the application layer register certain specific events with the event notifier. The agents define one or more of their target scenario with the event notifier during the registration process. The event notifier constantly checks for the occurrence of this pattern in the Interpreted database. The requests from the event notifier pass through to filter to check the unauthorized access of one entity's attributes by another.

4.3.3 Agent Deployment Block

ADB controls the agents and acts as a gateway for external agents to the context aware environment. The ADB is comprised of multiple agents each of them specialized in particular tasks regarding agent management. The ADB can be visualized as a cluster of agents working in a distributed fashion.

The agent platform forms the core of the ADB where agents are created or received. The agent platform provides a base for the agents to be identified and to carry out their operations. A detailed description of agents and the agent platforms is found in chapter 2. All the agents that conform to the standards of this platform can only be identified. There are four main agents in Agent Deployment Block.

1. Agent Handler
2. Control Agent
3. Resource Allocator agent
4. Mobility Management Agent

The agent handler performs the most important task of creating and dispersing agents into the ad hoc environment. The agent handler also interacts with the context analyzer to be notified about the presence of new entities by their own agents. If the entity does not carry any agents of its own, the agent handler spawns a new agent representing the new entity. This new agent will be featured with the attributes of the entity obtained from the context analyzer.

For further description of this entity, the new agent registers with the data storage block to identify the events and carry out the actions. Once the agent is dispatched the

control agent takes over the responsibility of agent management. Every agent records its actions with the control agent. The control agent also maintains conversations with the event handler regarding the states of dispersed agents.

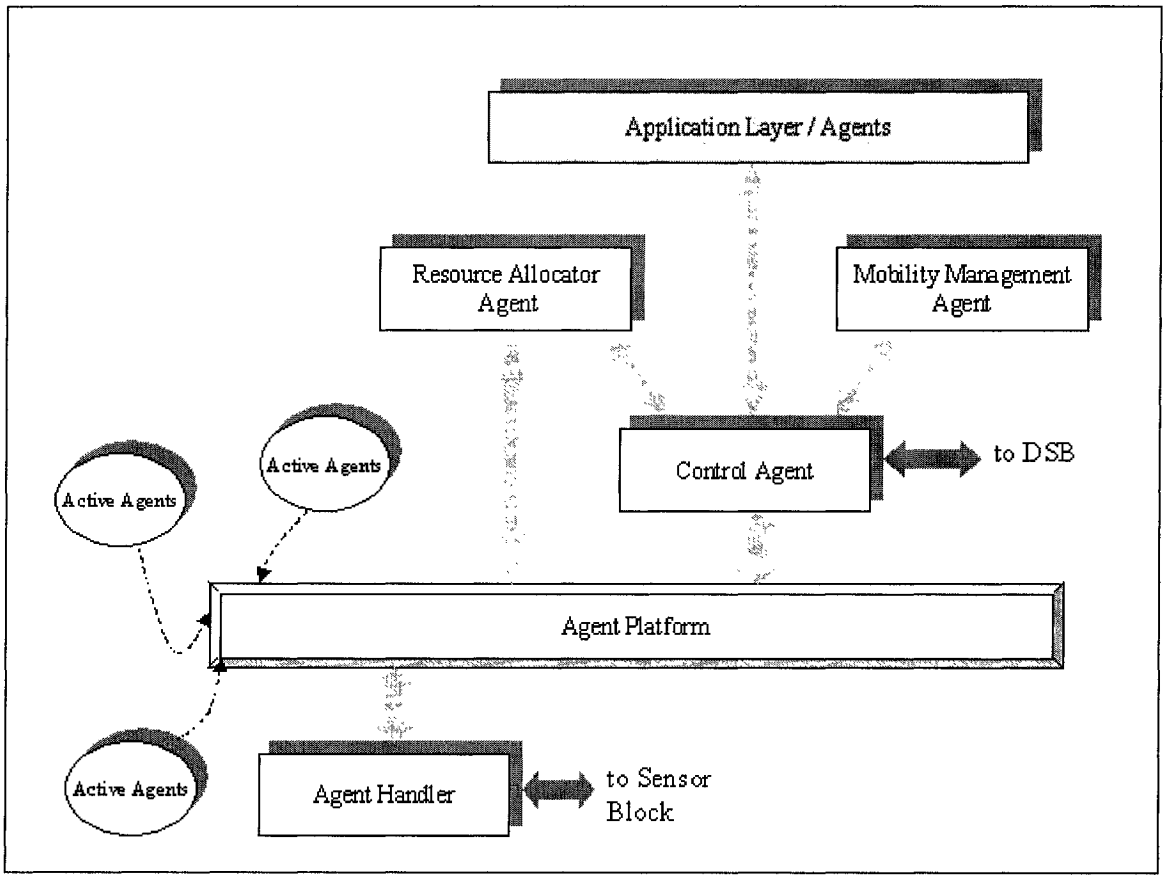

Figure 4.8: Agent Deployment Block

The life time of the agents are decided by the control agent by analyzing various factors like,

- Network Traffic
- Memory used by agent along with its data
- Resources held by agent

- Importance of agent, measured by the access frequency of agent data

The control agent is also responsible for interfacing with the external agents and enabling them to interact with local agents.

The resource allocator manages the list of resources in the environment. Resource refers to all the entities (Users and Services) present in the environment. The resource allocator maintains a map of resource allocations among the entities and their pending requests. It is the responsibility of the resource allocator to prevent resource allocation problems like indefinite waiting for resources and to provide appropriate messages to requester.

4.3.4 Query Handler

The main task of the query handler is to mediate between data and applications. The query handler identifies the request and provides with appropriate reply with or without data to the requesting module. Figure 4.9 depicts a modular view of the query handler. The query engine forms the heart of the query handler. It identifies the request and generates appropriate SQL or Object query. The request message is sent to the message parser to get the embedded attribute names in the message. The common messages parsed by the message parser are ACL, SQL or data streams from a socket.

The query engine maintains an index of entities and corresponding table names in case of local database, or object pointers in case of interpreted database. It matches the parsed elements with object or field names in the database to extract attribute values.

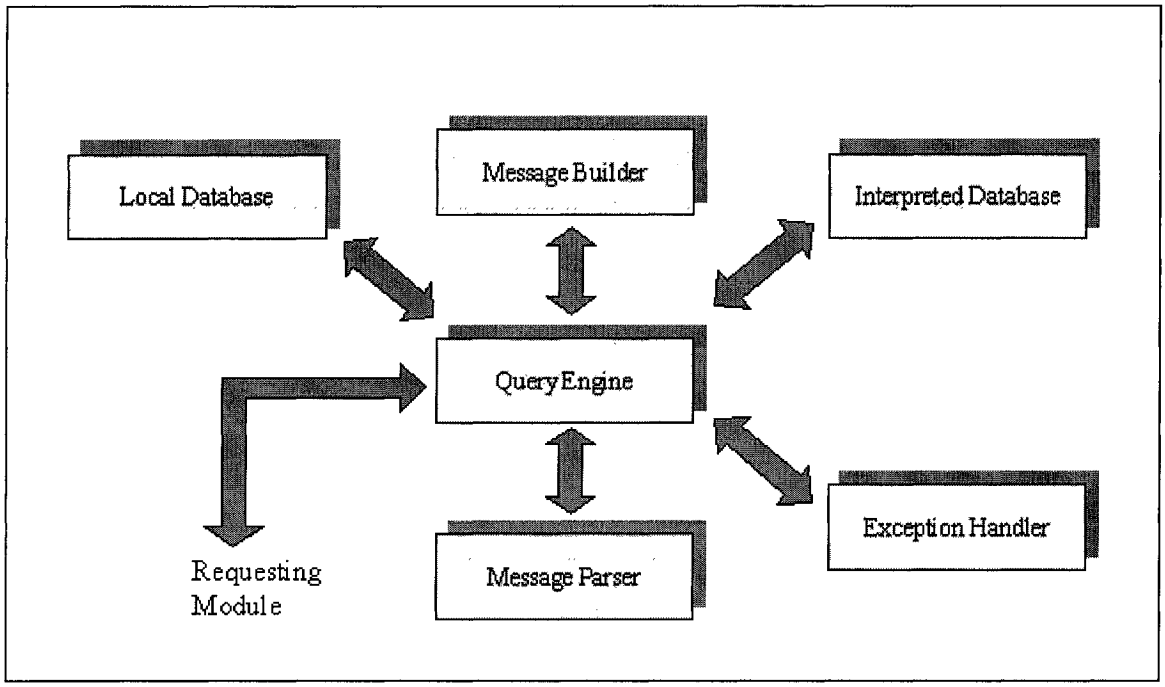

Figure 4.9: Query Handler

Once the appropriate values are obtained the message builder is used to convert the result into corresponding reply format demanded by the requesting module.

Query engine operates in multiple threads. Each thread is spawned whenever there is a new request. This thread is responsible for providing the solution for each query through interactions with various other modules. The query handler also provides safety mechanisms for threats like RAW or WAR that arises from a multithreaded environment. It employs semaphores and priorities for requests to overcome those problems. Once the request is being serviced, the query engine terminates that thread.

Another important task of the query handler is exception handling. Since the query handler performs plenty of database operations, it is prone to errors and exceptions arising from invalid requests or conversion errors. Exception handler captures these exceptions and maps them to application-specific error messages. The error message is

embedded into the reply message with appropriate attributes generating this error and sent back to the requester by query engine.

4.3.5 Filter

Filter module primarily is intended for providing security to context information. It handles two types of messages.

1. Information
2. Data requests

Information may be simple frequent presence announcements made by every entity, which does not have to bubble through all the modules every time it is sent. This type of message does not require a reply.

Data request is involved during access or modification in context information and the requester will be waiting for a reply. These are rather complicated since it involves authentication and demand security concern.

Since all the information is accessed via queries or ACL messages, the filter module screens those messages for authenticity. The filter maintains the preferences and policies of every entity. Policy governs the rights allotted to the different entities and also describes the preferences of the entities. Whenever an action is about to be performed on the context data, it is checked for conflicts with the policies of the related entities. The action may be just accessing the data or modifying the context information. The policies are applied to all the attributes subject to this action involving the requester and the target entity. Those attributes that pass the test are allowed to proceed further and those that have conflicts with the policies are sent back with appropriate messages.

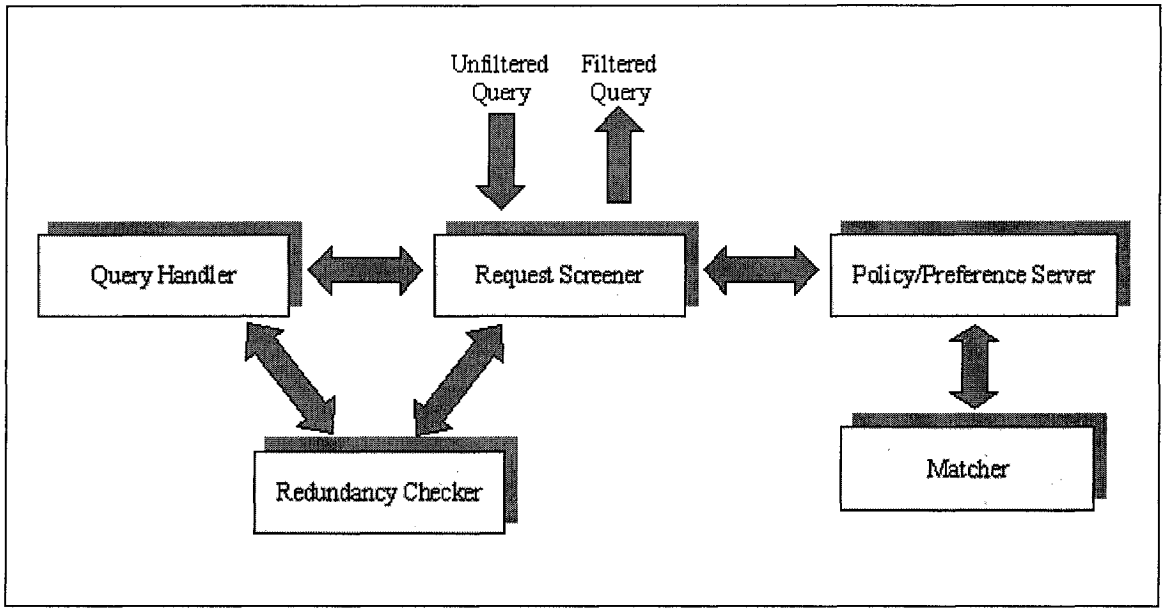

Figure 4.10: Filter Module

This is a block diagram of the filter module indicating the various sub structures in the filter. The unfiltered query is received by the request screener, which scans for the message type. If it is just information, then it is passed to redundancy checker. If the information is new, it passes the message unaltered to the request screener. If this is a message that is already received, the redundancy checker just updates the database with this information.

If the request is for data access, it is passed to the query handler for parsing. The request screener identifies the sender and the context attributes the sender is willing to access. The request screener then obtains the policies and preferences of the sender in accessing that particular data along with the preferences of the target entity if any. If there is a disagreement with one of their policies, the screener removes that attribute from the query and passes the rest. Once a set of qualified attributes are obtained, it is sent to the query handler to get them converted back to message format.

4.4 Layered Representation of CMS

This section introduces a different representation of CMS with respect to the major functionalities of the system. The CMS model can be viewed as a three-layer system in which each layer is devoted to a specific set of tasks.

The top and the bottom layer feed information into the middle layer, which is responsible for managing and securing this information. The lower layer is composed of physical sensors that acquire and process raw sensor data. This is fed to the middle layer where data is stored in appropriate formats (Context objects, XML profiles, database tuples etc). The upper layer uses this information to carry out tasks as well as derive context information from them and updates the middle level database with those derived values. Figure 4.11 represents the layered model that was just described. The subsequent sections explore each of these layers in better detail.

4.4.1 Context Sensing Layer

Being the lowest layer of the model, it interacts with physical components like sensor devices and gadgets. This layer is responsible for capturing low-level context data and converting them to application centric data. This conversion is performed by hardware devices themselves or by appropriate softwares associated with those devices. This layer also generates corresponding calls whenever a new change is detected. The sensor block from the CMS architecture constitutes the context sensing layer and the tasks performed by this block best describe the activities taking place in this layer.

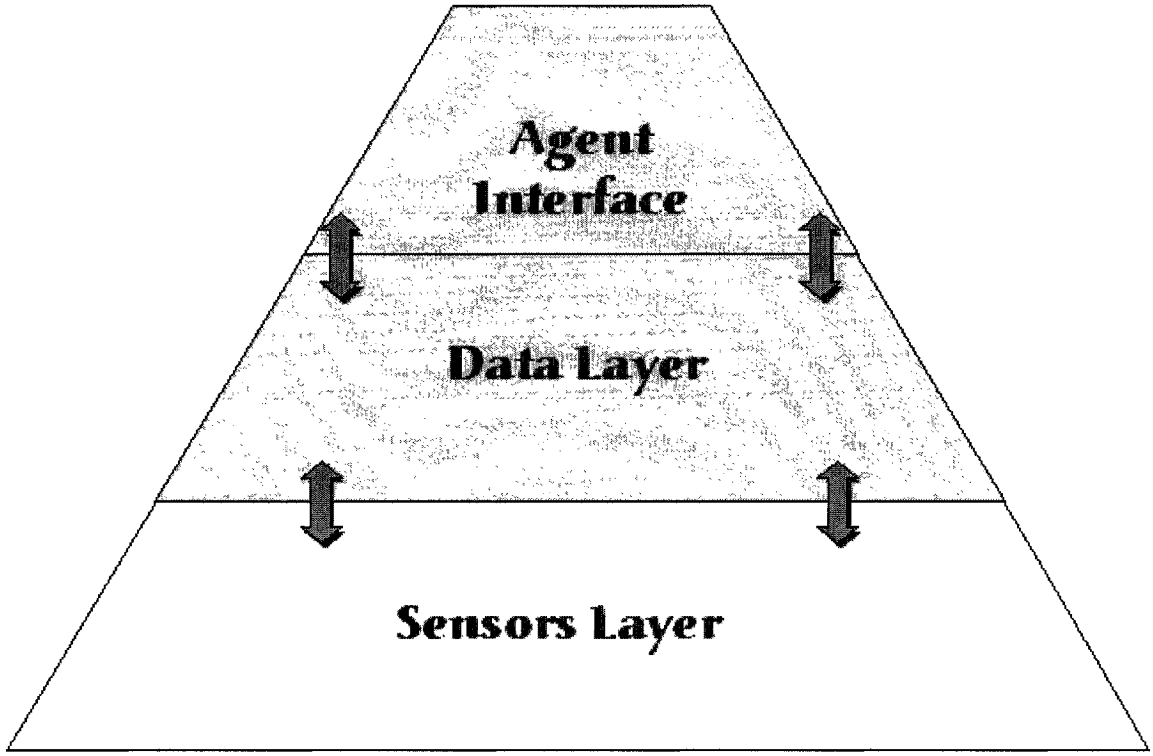

Figure 4.11: Layered Representation of CMS

4.4.2 Data maintenance Layer

The middle layer in this representation is the Data maintenance layer, which acts as the context information database. It supports the most common forms of data and data interpretations. It stores and manages different forms of context data and provides inter-conversions among them. Data Maintenance layer also incorporates the security issues relating to the accessing of context information by external entities. The middle layer provides support for various requests and reply formats involving multitude of applications. This information base forms a channel for the applications to take control and monitor the hardware components. This is a smart data storage that facilitates various data types and carries out conversations with applications requesting data.

4.4.3 Execution Layer

Execution layer is the agent layer or the application layer that forms the top most part of CMS. The execution layer uses the information from the data maintenance layer to achieve context awareness in the environment. All these activities are carried over by agents since they themselves are built upon this information base. The intelligence in data processing is integrated into this layer through agents that carry out activities arising from a context change. The execution layer captures soft context from higher level applications and reflects the change in the data maintenance layer whenever required. A variety of external applications that provide context information are identified and supported by this layer thereby creating an ad hoc environment. The execution layer may bubble its commands through the maintenance layer to control the sensors and hardware gadgets.

4.5 Design Issues

While modeling the CMS some key issues were identified relating to the characteristics and operation of the model. The following sub sections discuss those issues with respect to CMS model.

4.5.1 Versatility

Versatility has always been the focus of interest in ad hoc communication models due to diversity in devices that collaborate in the environment. The system, apart from being able to identify the device should also be aware of the techniques of communicating with that device. The CMS has to carry out negotiations with the devices to agree upon a particular QOS for every specific device. It gets complicated when the

preferences and policies of those devices come into practice. The main issue in this regard is the mode of communication with the terminal device. The ad hoc environment may be composed of devices ranging from a PDA with limited resources to a desktop having rich features and plenty of resources. The CMS must be able to capitalize on these capabilities of the target device in providing a service or collaborating with the device.

4.5.2 Security

The most interesting issue of any application based on an ad hoc environment is security, since the network is formed dynamically by relatively new entities. Therefore the entities are exposed to common network threats by anonymous nodes. There is always a trade off between security and ad hoc nature. When the security level increases it results in curbing the freedom of operation of the nodes and thereby denying the ad hoc nature. Thus the security system must be carefully analyzed and applied over the nodes or entities in the environment. The policies and preferences to some extent share the burden of providing the security for the host entities but do not ensure safe operation as there are still possibilities of those policies being corrupted or misused.

The CMS employs a three level security mechanism to provide the best security features to the environment.

The lowest or the physical level provides security by gathering information about the system configuration. It intercepts the data packets transmitted by the nodes for scrutiny and makes sure that the illegal or unauthorized requests are blocked from reaching other nodes in the network. For example the physical level may block suspicious data packets from reaching the policy server or the query handler.

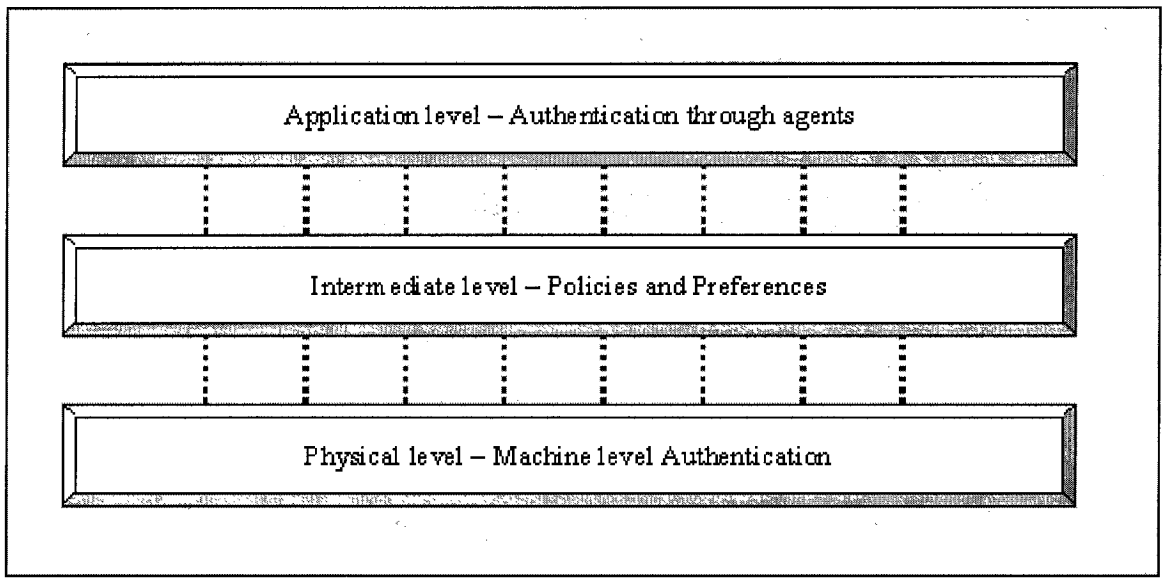

Figure 4.12: Security Implementation in CMS

The intermediate level performs policy authentication and avoid any access to resources that violate the policy rules drawn for each entity. These user-defined settings are dynamic unlike the physical level security and therefore are subject to frequent modifications.

The agents provide the security at the application level where it checks for the agent's identity before processing its requests. The most common problem handled by intermediate level is forging, where one agent disguises as another. The agent security checks the mobile agent code to see if it performs any illegal operation in the environment. Digital signatures are used when local agents in the CMS converse with each other. But when this is applied to external agents it shall tamper the dynamic nature of the environment.

4.5.3 Mobility

Agent mobility is one of the key issues that were encountered while trying to model CMS. Mobility does not deal with transporting agents alone but also their cargo. Agent mobility is not provided by some of the mobile platforms and hence in those cases CMS must employ its own mobility techniques in the agent deployment block. Transporting agent's cargo can create serious issues when the cargo contains multimedia documents, since these demand high bandwidth and heavy resources like disk space and complex applications to execute them. This can be costly when the target device is a hand-held, which does not support such high resources. The agent in such cases should migrate to the destination and stream the cargo if that is under acceptable limits. Agent cloning and streaming it through sockets is one alternate technique used by CMS for migrating agents in platforms that does not support agent mobility.

4.6 Summary

This chapter dealt with the core of this thesis, which is context management. The CMS architecture was broadly described with the organization of various blocks and how they contribute to achieve the goal. The features of CMS architecture were briefly discussed. Each of those blocks was described in detail to get a better insight on the components in each block. The usability and effectiveness of agents in the operation of CMS was identified. The agent-based communication between the various blocks was explained and how it reduced and how it simplified CMS design was analyzed. In a different perspective a layered representation of the CMS model was shown with the different blocks of the model grouped into these layers.

Based on this design some of the issues that arose during the design process and the way in which CMS reacted to it were explored. Among those issues security was dealt with more concern and different blocks in the CMS checked every possible entry into the system. The three-level security implementation was pictured to better explain the security features employed in CMS and finally the crucial capability of agents, mobility and its implications were discussed.

Chapter 5

Implementation and Results

This chapter discusses the prototype model that was implemented with the screenshots obtained during the different phases of the execution of CMS. It deals with mechanisms of implementing the various blocks in the CMS and how context information is being managed. The chapter explains how the various phases in the CMS are implemented and the tools that were used during this process. The feedbacks and the experience that was obtained from implementing this model are discussed. The performance of the model was also studied and the results are provided further in this chapter.

5.1 Scenarios

Some of the possible scenarios that take place in the room and how the system reacts to them are presented in the following sub sections. The scenarios involve two main entities, which are users and services and the activities that take place when they collaborate with one another. The mechanics of collaboration is beyond the scope of this thesis.

User/Service entering the room:

Figure 5.1 shows the various activities that take place when a new service or person is spotted. Firstly, the verification of tagId associated with the person is done. If the tag has a profile associated with it, then that profile is parsed and a service/presence agent is created based on the profile.

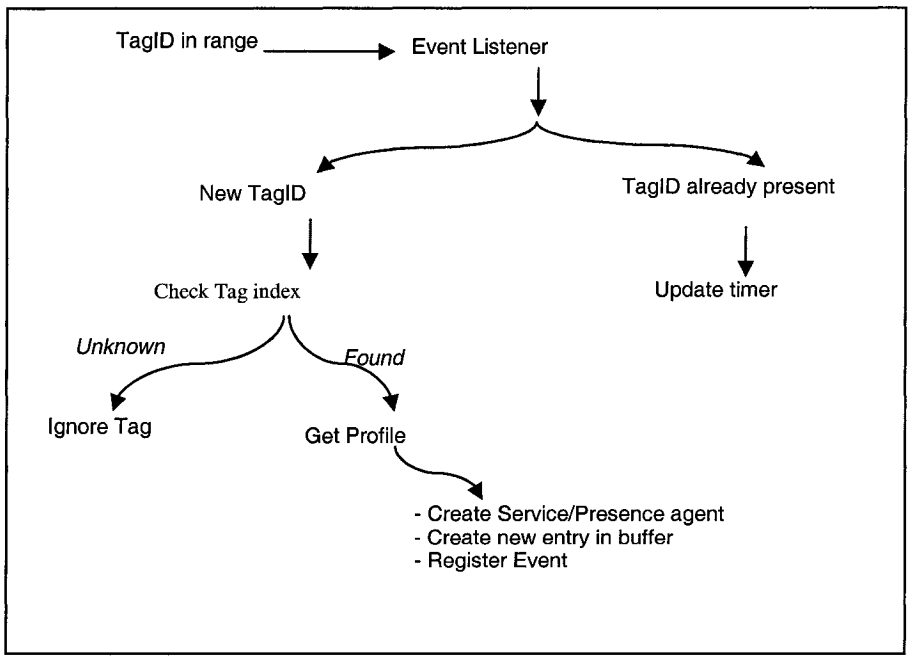

Figure 5.1: User entering the room

The agent is set to monitor the activities of the service or user and transmit the readings to the upper layers of the application. Any further conversations with the entity will pass through this agent. The service/presence agent will also be responsible for the changes in the state values of the context variables.

For software services, since they are not associated with any tags, the agent is already present and has to just register with the platform and then it acts similar to any other agents.

User/Service exiting the room:

In case of a user or service exiting the room, the event listener, which constantly monitors the tags, informs the corresponding wrapper agent about the absence of the tag.

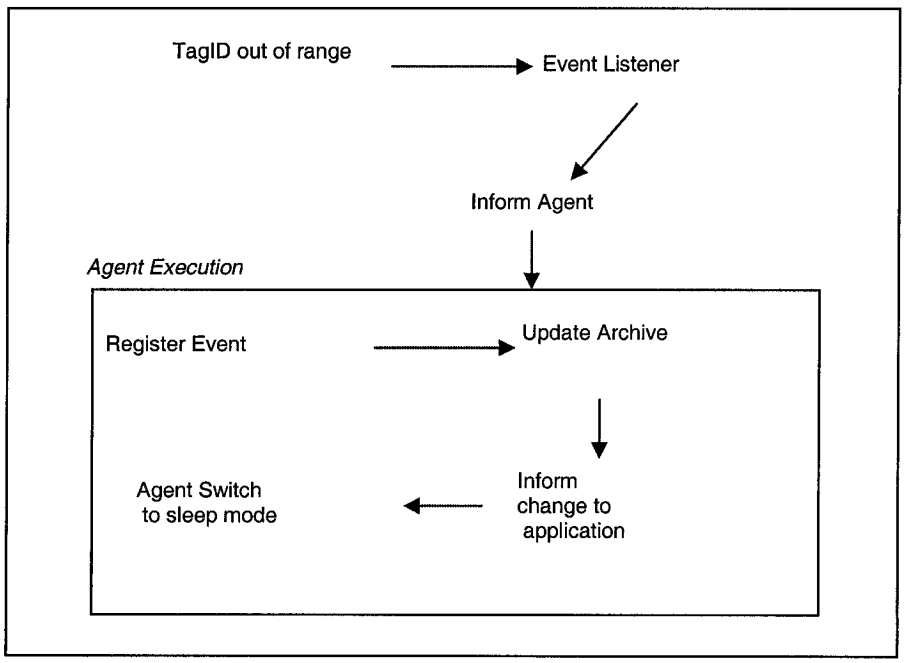

Figure 5.2: User leaving the room

The agent then carries out the process of registering this event and informing the upper layers about the change. Once the user/service has left, the agent switches to sleep mode instead of destroying itself.

User/Service performing an action:

The events performed by a user or service are also recorded. With respect to service, the events that could be possibly recorded are

- The service performing the requested task by the user
- The status of the service

With respect to the user,

- Request action
- The service accessed by the user
- Status of the user with time stamp

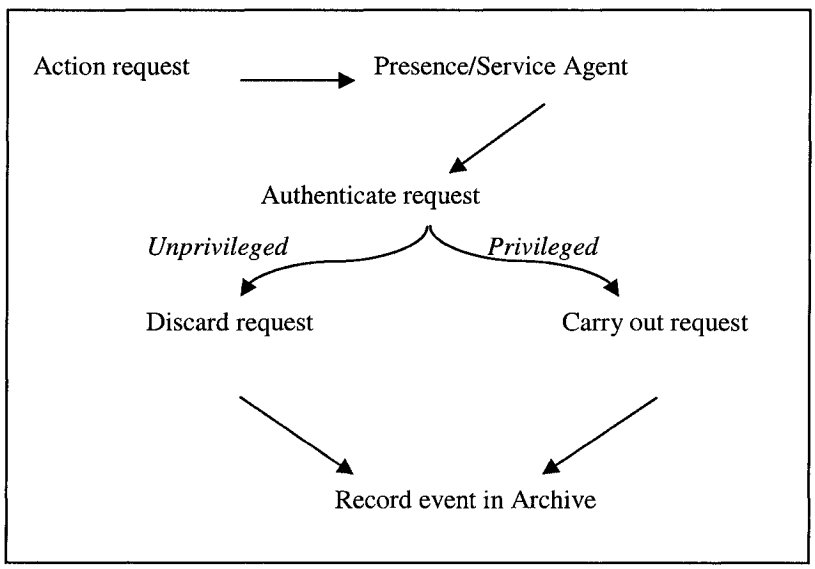

Figure 5.3: Action Performed by Service or Person

In this case both the service and the user are treated as similar entities. The difference occurs only in the status variables that are subject to change. The system just records the new value of the variable representing the object. The request is first authenticated and then the log is updated with corresponding results.

Figure 5.3 illustrates the sequence of actions that take place when a request for an action is placed either by a service or a person. The authentication is based on the policy settings of the entity. If the entity is privileged to perform the action, then the request is carried out otherwise it is discarded and an appropriate message is sent back to the requester. The result of the request is always stored in the archive irrespective of the outcome of the result.

5.2 Prototype Model

The implementation of the prototype model along with the snapshots that carry out the scenarios in the previous section will be discussed. The tools that are used during this process are also explained.

5.2.1 Implementation

The implementation of CMS prototype mainly focused on obtaining context information and managing it. To describe the prototype of CMS, the implementation of each of the different steps detailed in section 4.2 is explained.

Context Capture

The primary context information that was required is the presence of entity. The presence was obtained through radio frequency sensors by associating a tag with each

physical entity. Every tag has a UID (Unique Identification number) that is assigned to a particular entity to identify its properties. This UID or TagID is mapped to the entity's profile to obtain its description.

This is the detection technique used for physical objects in the active space. The identification of the tag in the room corresponds to the presence or absence of the entity associated with the tag. The object is said to be in the space as long as the sensors read the associated tag. Once a sensor does not find a particular tag in the space, it assumes the absence of that entity.

The second type, which is the soft capture, is used to detect the network connectivity of client devices to the ad hoc network. Specific agents in the network that continuously check for the new device connections capture the device properties and the services offered by the connected device. Capturing user activities also is a part of soft sensing. In this prototype, the user status is captured along with the time stamp. The different status messages available to the users are busy and available. Similarly from a service point of view, for example, the status messages of a printer are available, busy, out of paper and disconnected.

Context Description

For sharing and using the services in the ad hoc network, the environment must be aware of the identity and features of every person. In an ad hoc network the users enter and leave the network dynamically and hence the user identification also has to be dynamic rather than having a pre configuration for every user. The agent that represents each person takes over this responsibility. This personal or presence agent deposits an

XML file describing the features of the owner in a public protected folder. By this method any person who is new to the environment is also managed the same way as other inmates. The formation of the profile can also be done manually by depositing a profile in the same folder when an entity is assigned a tag.

In case of the software services, the agent representing the software volunteers to announce itself to the network about its availability, as there is no tag association with such services. The software services provided either by the room entity or by one of the users willing to share a service with others in the room. Once the service is identified, the profile of the service is parsed to get the description of the service and the ways to communicate with it. The parsed service description can also be obtained on request from the service agent. The prototype was tested for two services, a printer as a hardware service and a PDF writer as a software service that is provided by one of the users.

Policies were employed to describe the user's preferences. This forms a more complex context description of a person that is not provided by the basic profile as well as adds security to context data. Every user has a simple set of policies to describe some sample scenarios, which we shall see in the later sections.

Trigger action

Agents in the prototype generally carry out actions. They activate a service by providing a user interface to the requester based on certain conditions like matching of policies provided for each user.

Whenever there is a request for a service by an agent, the service agent migrates to the requester's device and displays an appropriate GUI for the user. This action is

independent of the device type of the user as the agent automatically performs the QOS for the target device. The agent accepts the input from the user and carries out the task in the same location if possible or on a different location. For example, when a user requests a printing service, the service agent migrates to the user's device and accepts the file to be printed. Since the user device may not have appropriate device drivers for the printer, the service agent transports this file to another device with the drivers and gets this file printed.

The agents monitor and manage the context attributes to perform the matching through context objects. Figure 5.4 shows a sample context object of a person.

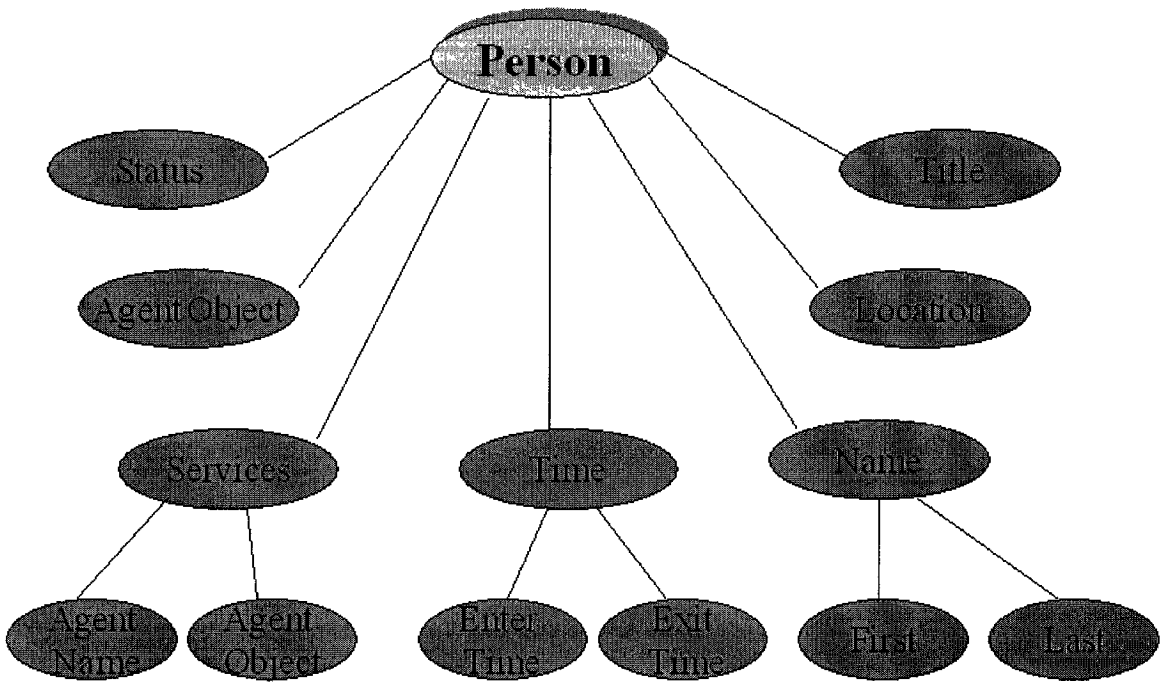

Figure 5.4: Context Object of a person

The agent automatically creates the context object for each entity and the object gets exchanged between agents when there is a transaction. The context object remains active until the agent terminates it after transferring the object contents to the local

database. The local database contains the profile of the entity during its last visit and is retrieved back during the next visit. An exclusive policy server was integrated into the prototype to manage the policies. The details of policy management are not handled in this chapter as it deviates from the intended target.

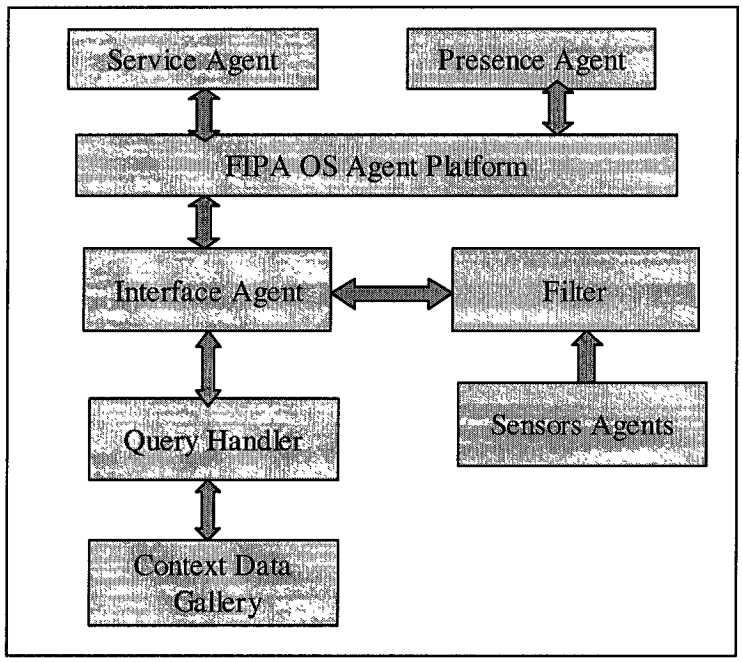

Figure 5.5: Overall model of the Prototype

The above figure represents the overall model of the prototype. The different blocks of the prototype are shown with the interaction among the blocks. The sensor agents provide context data to the interface agent through the filter. The interface agent populates and maintains the context data gallery with the information obtained from sensors as well as other agents. The interface agent is responsible for creating and managing service and presence agents by triggering appropriate actions based on the context data. The following section provides a detailed explanation on this modular view of the prototype.

5.2.2 Modular View of the Prototype

This section describes the individual modules in the prototype in a better detail. It realizes the various blocks in CMS that helped in analyzing the CMS architecture.

Sensor Block

The sensor block accommodates two sensors. The RF sensor kit is of the hardware sensor type and the network watcher of software sensor type. While the RF sensor monitors the physical presence of entities, the network watcher monitors the connectivity of the existing and new devices to the network.

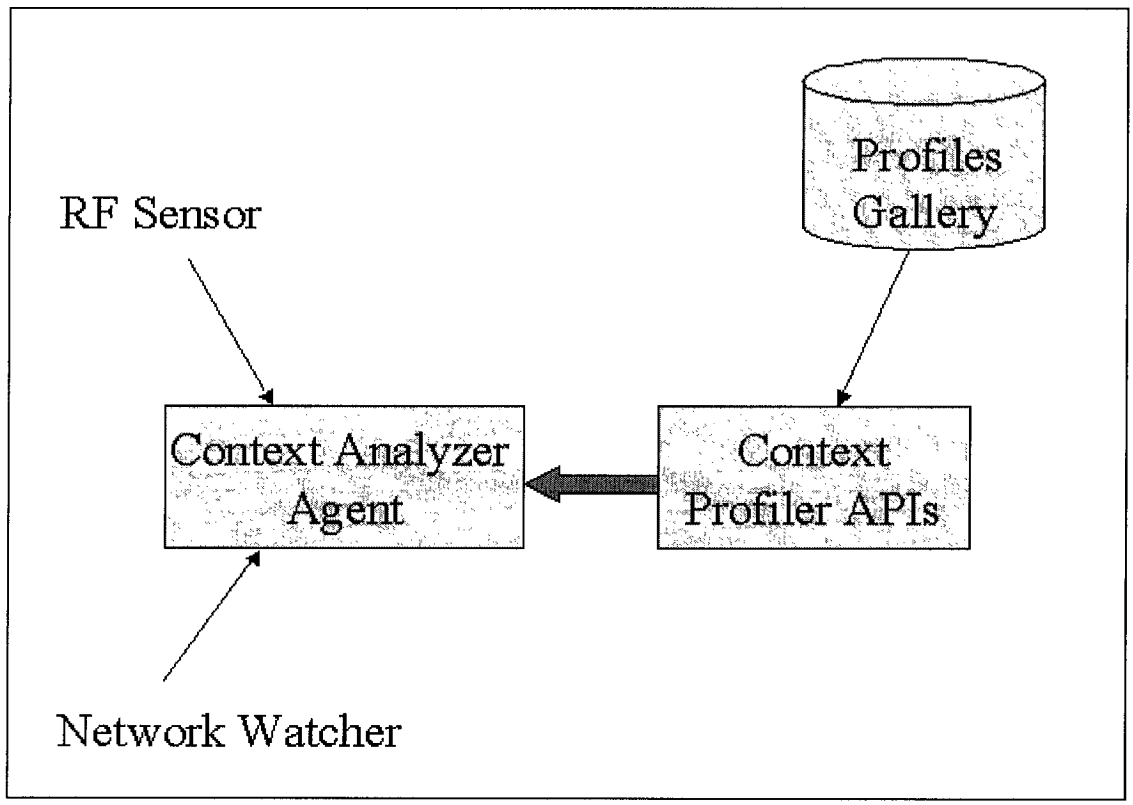

Figure 5.6: Sensor Block Implementation

The sensor controller interfaces with the hardware component of the RF sensor to stream presence information of entities and context analyzer agent converts them into hashtables by matching it with the preset profiles of known entities and default profile for the unknowns. These profiles are stored in profiles gallery and the context profiler parses these profiles and the values are available through APIs to the context analyzer. Context analyzer agent provides the communication channel for exchanging context data between other modules in the implementation

Data Storage Block and Agent Deployment Block

The agent deployment block and the data storage block can be coupled together as the agents control the context data storage and retrieval. The Interface agent acts as the control agent in creating and controlling service and presence agents. Whenever the presence information of a new agent is informed to the Interface agent, it identifies the entity type and creates appropriate agent. The interface agent also manages the object database containing the context objects of the presence and service agents. Interface agent maintains a temporary profile gallery that acts as a local database. The profiles in this gallery represent the corresponding entity during its period of stay. The service or the presence agent when required to move to another device, requests the mobility management agent to perform object migration as FIPA-OS does not feature this capability.

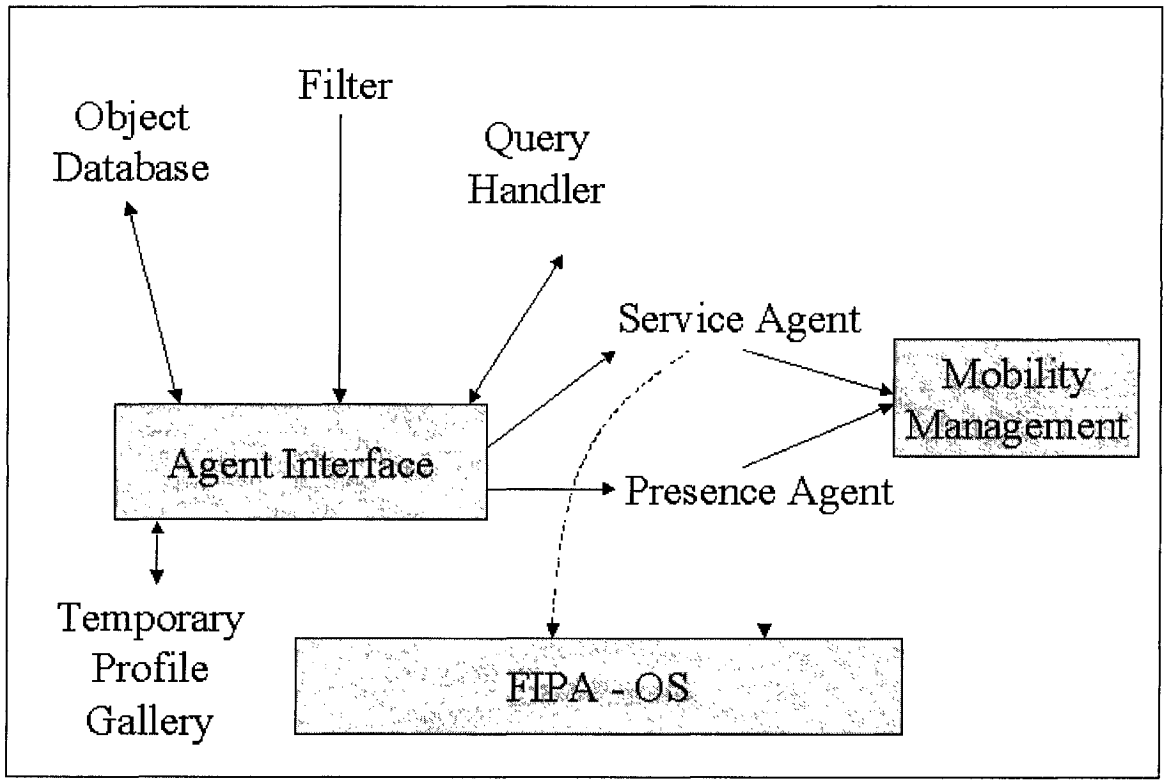

Figure 5.7: DSB and ADB Implementation

Filter Module

The implemented filter module prevents the redundant presence information from reaching the DSB. The context analyzer agent communicates with the filter agent about presence status of every entity. Filter agent checks with the temporary profile gallery for the presence of the entity. If the entity is already present the agent updates the TTL(Time to Live) attribute of the entity otherwise if the entity is new to the environment, it passes the information to the agent interface.

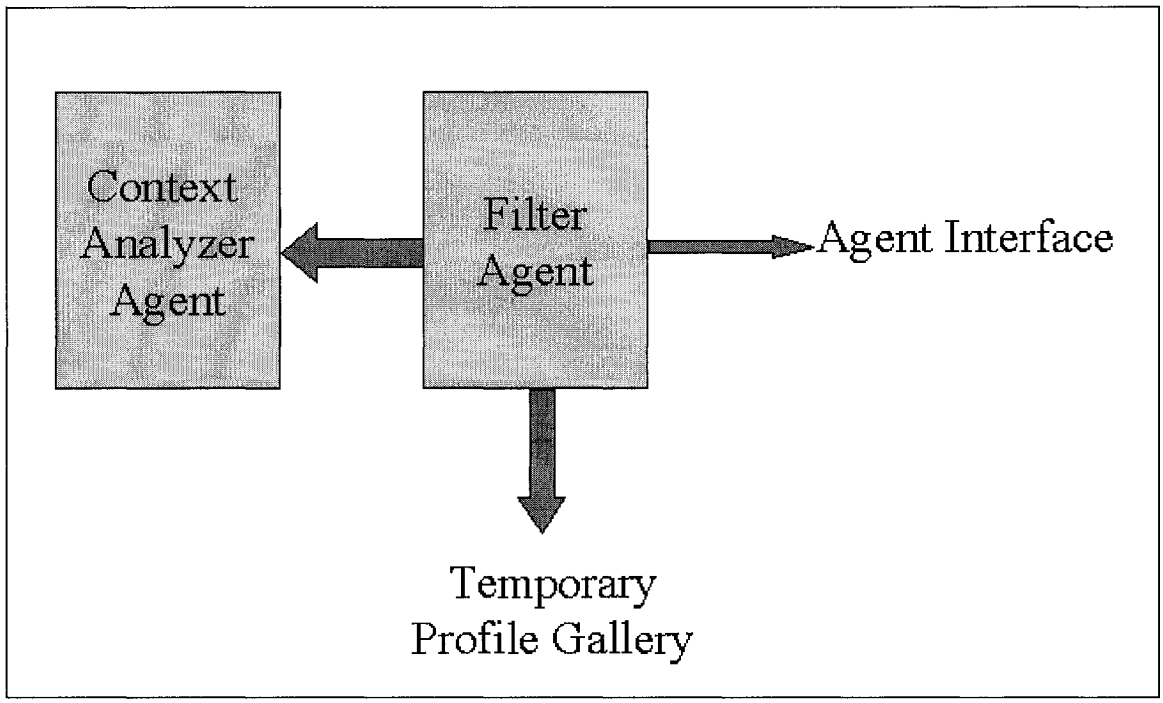

Figure 5.8: Filter Agent Implementation

Agent Communication

The following diagram shows how the agent communication is carried out when an action takes place.

When a data is received from a sensor, the context analyzer agent identifies it and passes to the filter agent. The filter agent communicates with the query agent to obtain the presence information of the entity. The filter agent updates the TTL in case of the existing entities or informs the Interface agent when otherwise. The Interface agent creates the context object and spawns presence or service agent.

When there is a request for using a service or just a request for obtaining information regarding any entity, the corresponding service or presence agent receives

and processes the request. The agent also updates the archive with the requester information and request type by communicating with the query handler.

In case of a move request, the agent requests the mobility management agent to move its object to the destination location. The following figure portrays the communication that takes place between the different agents in the prototype.

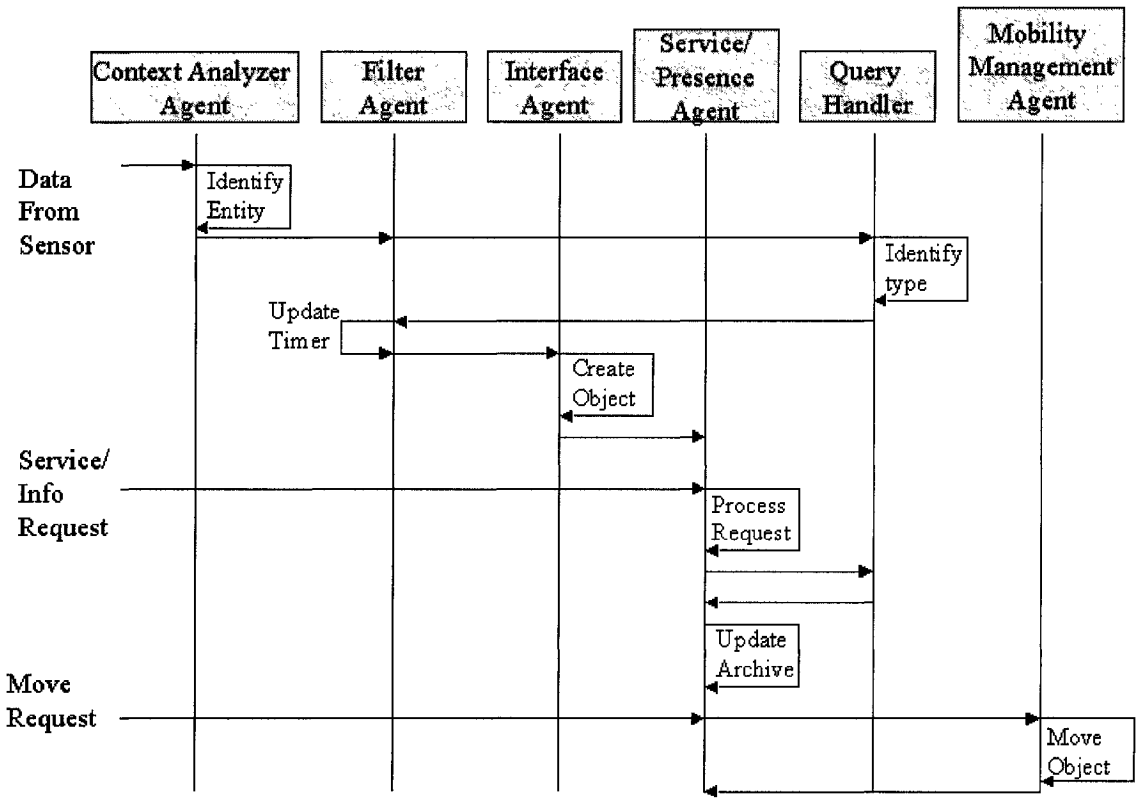

Figure 5.9: Communication between different agents in the prototype

5.2.3 Tools Used

The prototype of this model was implemented and tested in JAVA since it is system independent and ensures adaptability when comes to mobile devices. Service and Presence agents are created and deployed using FIPA-OS. For hand-held devices, a special version of FIPA-OS called Micro FIPA-OS was used which is a compact version

containing only the basic features. Since the agents use a common platform, all the agents must follow the FIPA standards if they need to communicate with one another or move from one platform to another platform.

Mantis Kit from www.rfcode.com was used for tag sensing. There is approximately a maximum of ±5 feet error in calculating the distance of a tag. This directly affects the precision in finding the tag if it is inside or outside the room. Therefore we set a time limit of 20 seconds between the emissions of beacon. If the beacon is not heard for more than 20 seconds then the tag is considered to be out of range. The major issue that arises is the interface and communication among various types of services, because there is not a common language or protocol that is understood by all services. X10 [56] is one such communication language used for electrical appliances to talk with each other. The prototype implementation basically monitors, manages and stores contextual parameters. The entities considered are services and people in a physical room space. There can be more than one room space, where the users and services in these rooms can collaborate with each other.

XML is used as encoding language for entity profiles since it provides portability and flexibility. The profiles of the entities are represented in XML also the object backup is performed after converting them to XML and storing them as XML files. For implementing local database, MS-ACCESS is used along with JDBC for storing and retrieving data. Though a simple RDBMS, ACCESS proved easy for implementing and performing modifications during testing and documentation.

5.2.4 Snapshots and Code Snippets

In this section the CMS prototype is explained using the snapshots taken during the execution of CMS and some of the code snippets that carries out certain specific tasks.

The snapshots shall be discussed based on the scenarios presented in previous section. The following picture shows the screenshot of the log window that displays the activities that takes place when a user enters the room and leaves after some time. The first line of the screenshot displays the time and the distance at which the user was spotted. The name of the user (Mr. Gray) is obtained from the local database after mapping the tagID of the user with the profiles. The context information from the local database is displayed as a hashtable containing the basic and cached attributes.

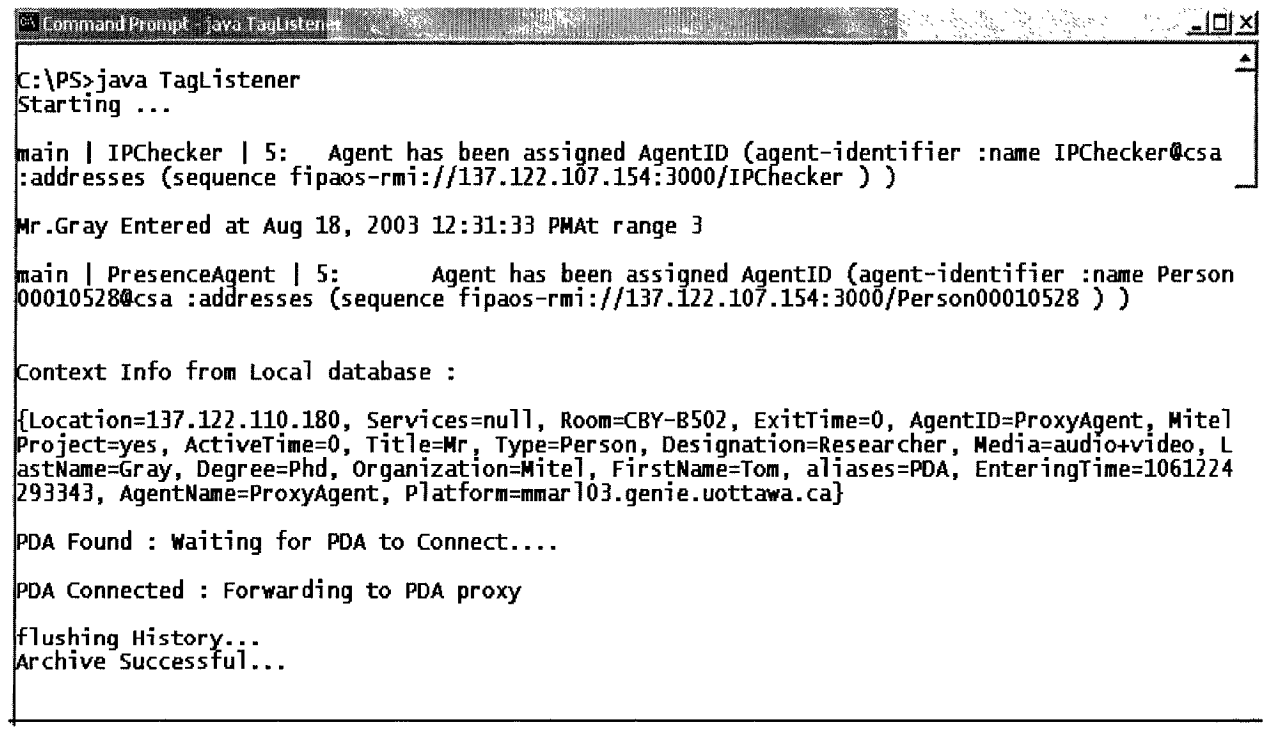

Figure 5.10: Person entering with PDA

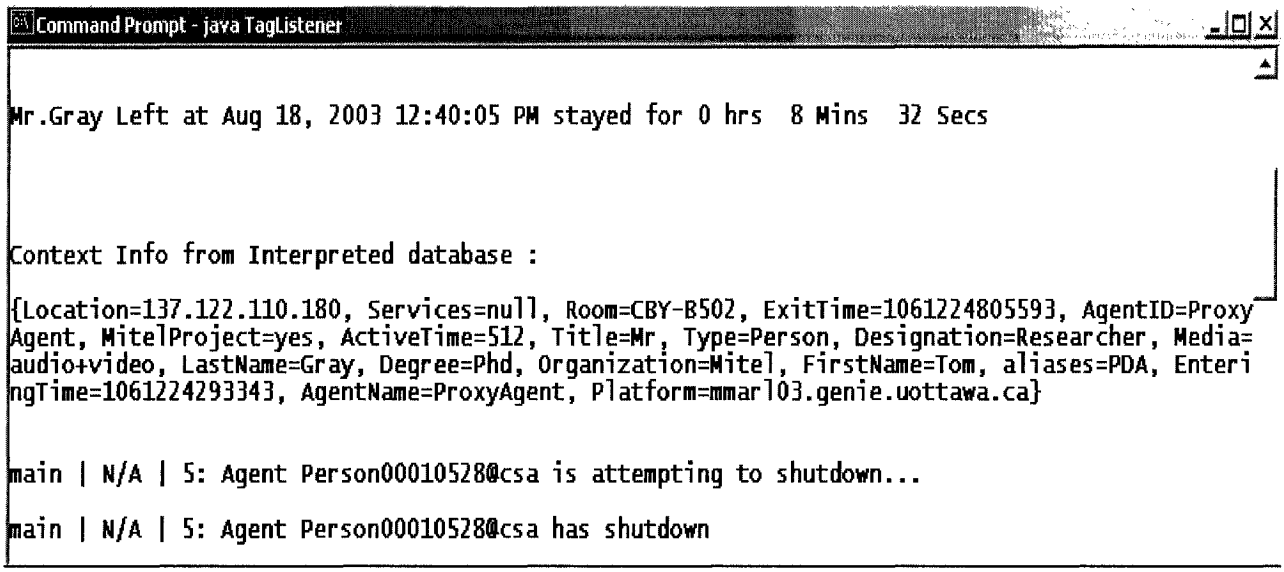

Figure 5.11: Person leaves with PDA

The cached attributes in this case are the platform name and the Media that was taken from the previous visit of *Mr. Gray*. Once the user connects his PDA to the network, the message PDA found is displayed. Since the PDA cannot support some of the features the context object of the PDA is forwarded to the PDA proxy for further actions.

The agent by the name *Person00010528* is created and this agent is responsible for creating the context object for this user. The following code snippets show how an agent is spawned and context object is created.

```
149     else{//It its a Person
150
151         _type = new String("Person");
152         pos = tempElements.indexOf(new String("LastName"));
153         System.out.println();
154         //Display a simple message about the person
155         System.out.println(tempValues.elementAt(pos-2)+" "+
156                            tempValues.elementAt(pos)+" Entered at "+
157                            (new java.util.Date()).toLocaleString()+
158                            "At range "+tidinf.getrange());
159
160         //Add the tagID of the Person as an index to the PrA
161         PrAlistIndex.add(tidinf.gettagID());
162
163         //Create a Presence agent with TagID as agent name and add it to the PrA list
164         PresenceAgent ph = new PresenceAgent("/fipaos/profiles/platform.profile",
165                                              new String("Person"+tidinf.gettagID()),
166                                              "FIPA-OS",tempElements,tempValues );
167         PrAlist.add(ph);
168
169
170         //System.out.println(PrAlist);
171         addToDb(tempElements,tempValues,tidinf.gettagID(),"Person");
172
173
174     }
```

Figure 5.12: Code snippet that creates agent and updates interpreted database.

The presence agent for the user is created with the attributes obtained from local database and the agent is added to the list of currently active agents. Once the agent is created, the interpreted database is updated by the method *putBackXML()*. The agent now possesses the context object and is responsible for updating and maintaining it. The following snippet shows the creation of context object.

tempElements and *tempValues* form the attribute value pairs in the context object that are obtained from parsing the XML file from the local database.

```
112         //Parse the XML
113         xpi.initParser(PROFILES_PATH+fileString);
114
115         //Store the parsed values in Vectors
116         tempElements = xpi.getElementsList();
117         tempValues = xpi.getValuesList();
118
119
120         //Modify Values in Context Object
121         tempValues.set(xpi.getElementsList().indexOf(new String("EnteringTime")),
122                        new Long(new java.util.Date().getTime()));
123         tempValues.set(xpi.getElementsList().indexOf(new String("ExitTime")),new Long(0));
124
```

Figure 5.13: Creation of Context Object.

Once the user is found in the network, the presence is archived. But before doing so, the archive is checked for entries that are more than 15 days old. The entries that are 15 days or older are removed from the archive and the message "flushing history" indicates this action.

When the user leaves the room, the information is logged with username and exit time. The interpreted database contains additional information about the user exit time and active time. Moreover the platform value and the media value of the user are updated. After storing the updated values back to the local database, the agent shuts down. Figure 5.9 shows the screenshot of the user's archive over a period of time. The enter time and exit time of the user are archived along with the status of the user at a particular time. The status value can be interpreted as follows,

0 – The user is available for interaction

1 – The user is busy

2 – The user is not at his desk

3 – The user will be there for a few minutes

IndexNo	Date	Services	EnterTime	ExitTime	ActiveTime	Status	Service Access	Access Time
29	1.0553496E+12	null	1.0553495306E+12	1.0553496709E+12	1403	0	HP LaserJet	1.0553495681E+12
36	1.0554467E+12	null	1.0554467068E+12	1.0554478320E+12	1125	1	HP LaserJet	1.0554475366E+12
38	1.0553682E+12	null	1.0553681137E+12	1.0553683160E+12	46	0		
39	1.0553683E+12	null	1.0553682693E+12	1.0553683830E+12	46	1		
40	1.0553684E+12	null	1.0553683306E+12	1.0553683837E+12	46	1		
41	1.0553684E+12	null	1.0553683401E+12	1.0553683449E+12	49	0		
42	1.0553688E+12	null	1.0553687443E+12	1.0553687900E+12	46	2		
43	1.0553688E+12	null	1.0553685816E+12	1.0553686264E+12	46	0		
44	1.0553689E+12	null	1.0553686873E+12	1.0553686954E+12	49	3		
45	1.0553690E+12	null	1.0553689587E+12	1.0553690093E+12	49	0		
46	1.0553691E+12	null	1.0553690320E+12	1.0553690781E+12	46	2		

Figure 5.14: User Archive

These values are supplied to the upper layers that require this context information for display purposes. Numbering the status proved to be a better option when it comes to deriving smart context like the automatic calculation of user status.

The snapshot in Figure 5.15 represents a similar activity when a service enters and leaves the environment. The context object contains the properties of printer that will be required by the upper layers that request this data. The service agent that performs the request matching takes these properties to provide QOS to the requester. The following piece of code represents the activities that take place when a service or a person leaves the room. The service or person that is identified from the tag and the corresponding agent is informed about the absence. The agent updates the local database and shuts down.

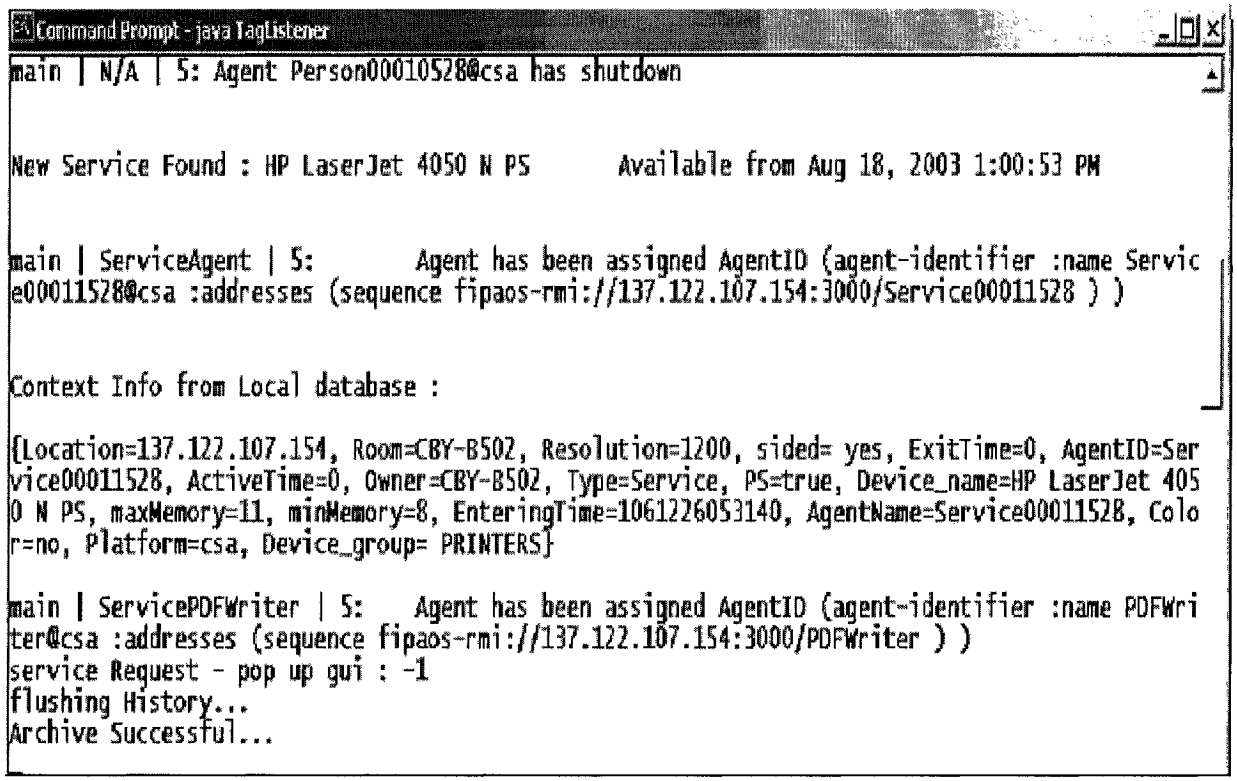

Figure 5.15: Service Enter

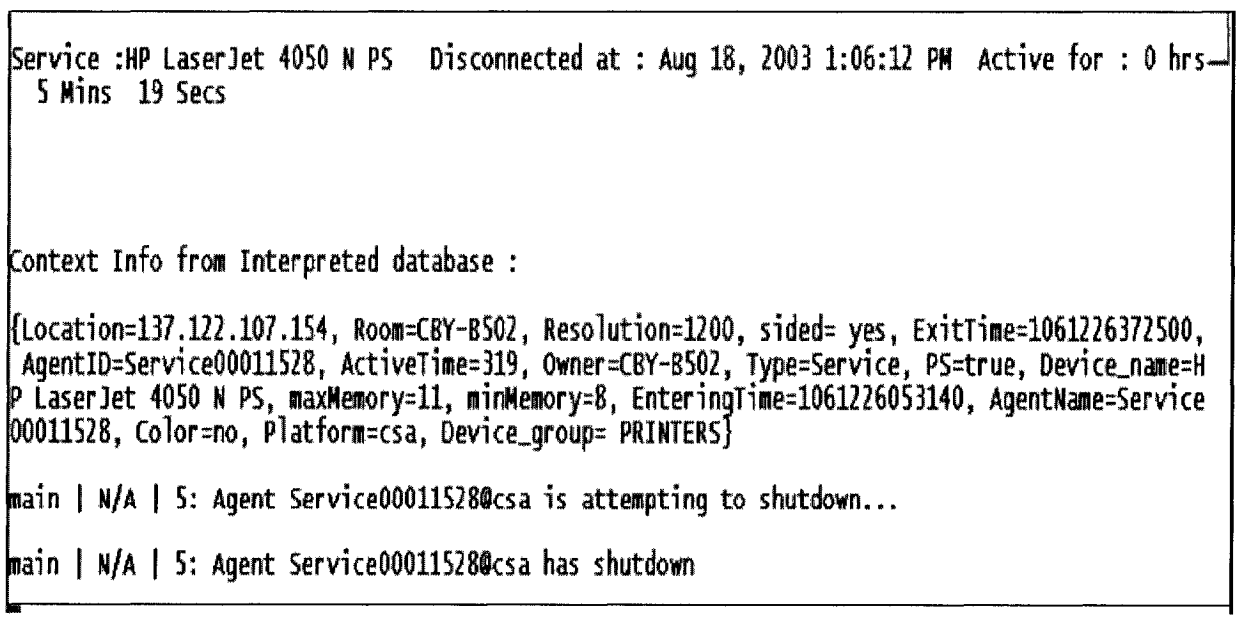

Figure 5.16: Service leave

```
//Check if the tag belongs to a service or a person
if(xpi.getValuesList().elementAt(xpi.getElementsList().indexOf(new String("Type"))).equals("S

    //Display a simple message about tag absence
    System.out.println();System.out.println();

    System.out.print("Service :"+ xpi.getValuesList().elementAt(xpi.getElementsList().indexOf
    System.out.println((new java.util.Date()).toLocaleString()+"  Active for : "+hrs+" hrs  "
    System.out.println();System.out.println();
    //Remove the Service agent from the current list of agents and notify the change
    ServiceAgent sa = (ServiceAgent)(SAlist.elementAt(SAlistIndex.indexOf(tidinfo.gettagID())
    sa.informAbsence(tempElements,tempValues);
    SAlist.remove(sa);
    SAlistIndex.remove(tidinfo.gettagID());
    updateDB("",tidinfo.gettagID(),"Service",ext,new Long(activeTime));
}
else{//if the tag belongs to a person

    System.out.println();System.out.println();
    //Display a simple message about tag absence
    System.out.print(tempValues.elementAt(tempElements.indexOf(new String("Title")))+"."+ tem
    System.out.println((new java.util.Date()).toLocaleString()+" stayed for "+hrs+" hrs  "+mi
    System.out.println();System.out.println();
    //Remove the Presence agent from the current list of agents and notify the change
    PresenceAgent pa = (PresenceAgent)(PrAlist.elementAt(PrAlistIndex.indexOf(tidinfo.gettagI
    pa.informAbsence(tempElements,tempValues);
    PrAlist.remove(pa);
    PrAlistIndex.remove(tidinfo.gettagID());
    updateDB("",tidinfo.gettagID(),"other",ext,new Long(activeTime));
}
```

Figure 5.17: User/Service leaving the room

Once the service agent migrates to the user terminal, it displays a GUI to get input from the user to carry out its operation. The following is a screenshot taken after printer agent migrates to the user terminal and waits for the user to select a file to be printed.

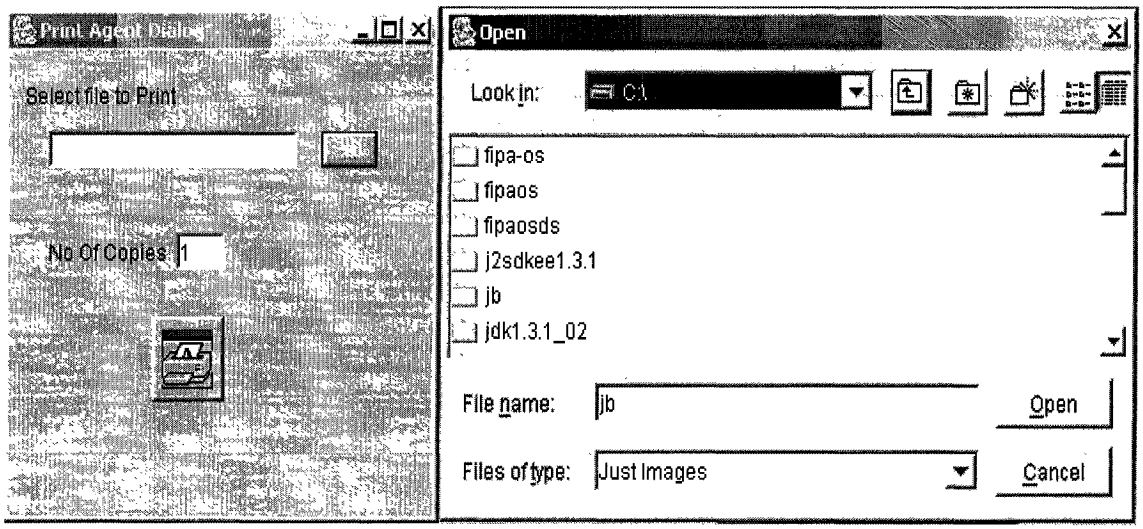

Figure 5.18: Print Agent Dialog

This is a simple GUI that takes in as input, the file name and the number of copies to be printed. An exclusive print agent that needs more information to perform QOS operations can also be served since the prototype maintains an extensive list of printer attributes.

5.2.5 Integration with mobile ad hoc communication project

The CMS provides the context-awareness feature as a part of the ad hoc communication project. The central idea of this project is to bring various types of users and services together into a network where they can collaborate with one another and share services in the network.

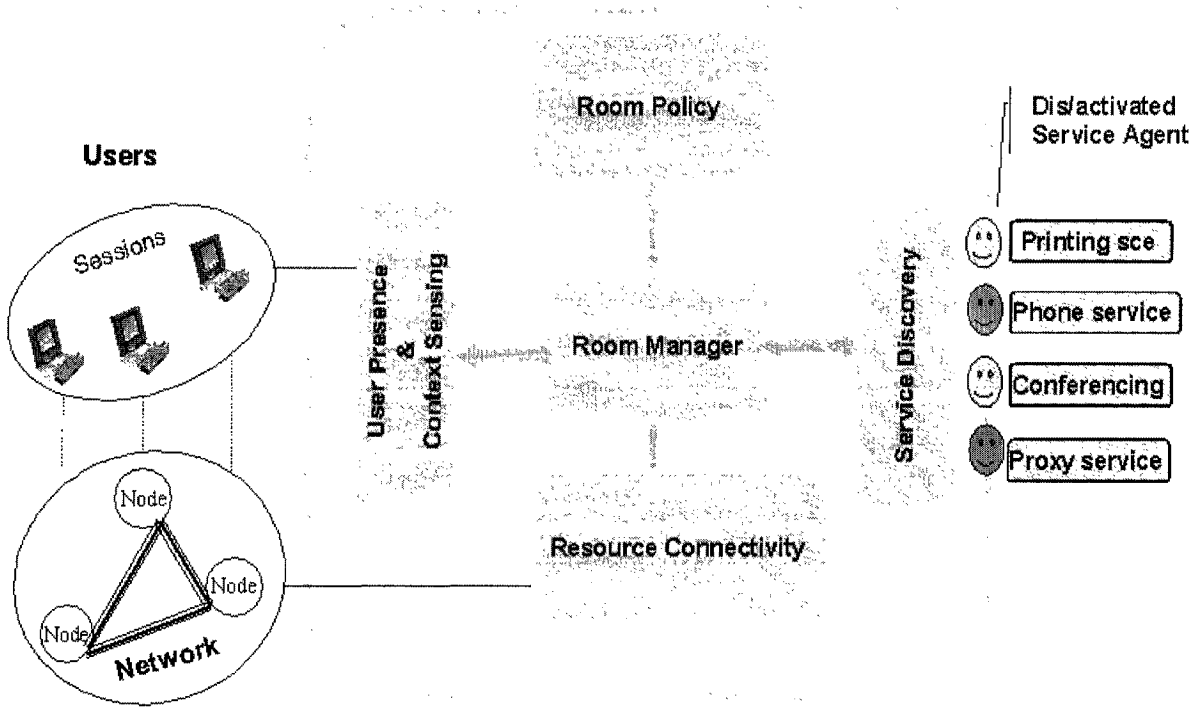

Figure 5.19: Context awareness as a part of Ad hoc Communication Project

The above figure represents the ad hoc communication project with the room manager as the central controller. The main purpose of this project is to enable collaboration between different entities, which are users and services. The users and services join and leave the room in a dynamic fashion. The users may or may not carry personal devices such as laptops or PDAs. The presence of these entities is identified spontaneously and appropriate tools for collaborating with environment are supplied to the client devices incase they do not posses the required tools. The underlying network can be wired or wireless depending upon the devices that are connecting to it. Every privileged user is made aware of the services and other users in the room after passing policy tests. The users are allowed to share their own services and use the available authorized services when they need and this is taken care by an exclusive service

discovery module. The service discovery provides the suitable service matching the request from the client in the same room or from other similar rooms. Once the target devices are identified by their capabilities, communication session is formed between them and monitored constantly. This session is managed by using the context information of the room and that of the participating entities. The resource connectivity module is responsible for creating and managing the communication sessions between the entities. It provides session management for the user – service or user - user interaction.

Every user, after connecting to the ad hoc network will be provided with a GUI that displays the other users and services in the network with which the user can collaborate. These services and user lists are displayed after performing policy checks for the users and the services. If the user wants to use the services, he/she has to just select the service and click on submit. The request goes to appropriate service agent and the agent migrates to the users terminal to perform further operations.

The conference tool interface is another part of ad hoc collaboration project that is not concerned with this thesis. But the conference agent request for context objects from CMS while providing interaction between two or more users.

Role of CMS in Ad hoc Collaboration System

The CMS provides the context sensing and the user presence information to the room manager. The CMS encompasses all the sensors in the environment that provide context information about the entities in the room. The CMS gathers that information and converts that into application understandable format and feed the context data to other modules upon request. The CMS can be considered as the lowest level in the system that

is hidden to the user who is interacting with the applications in the upper layers. The CMS identifies the resources and their attributes uniquely and supply this information to the service discovery and resource connectivity modules through the room manager. The agents in CMS interact with other similar agents in ad hoc collaboration system in exchanging context information. In general, CMS can be considered as a smart information base for the system that manages and delivers context information whenever required.

The room manager couples the context information from CMS with the service discovery mechanisms along with the policies of the participating entities in the room to enable collaboration among the people and services.

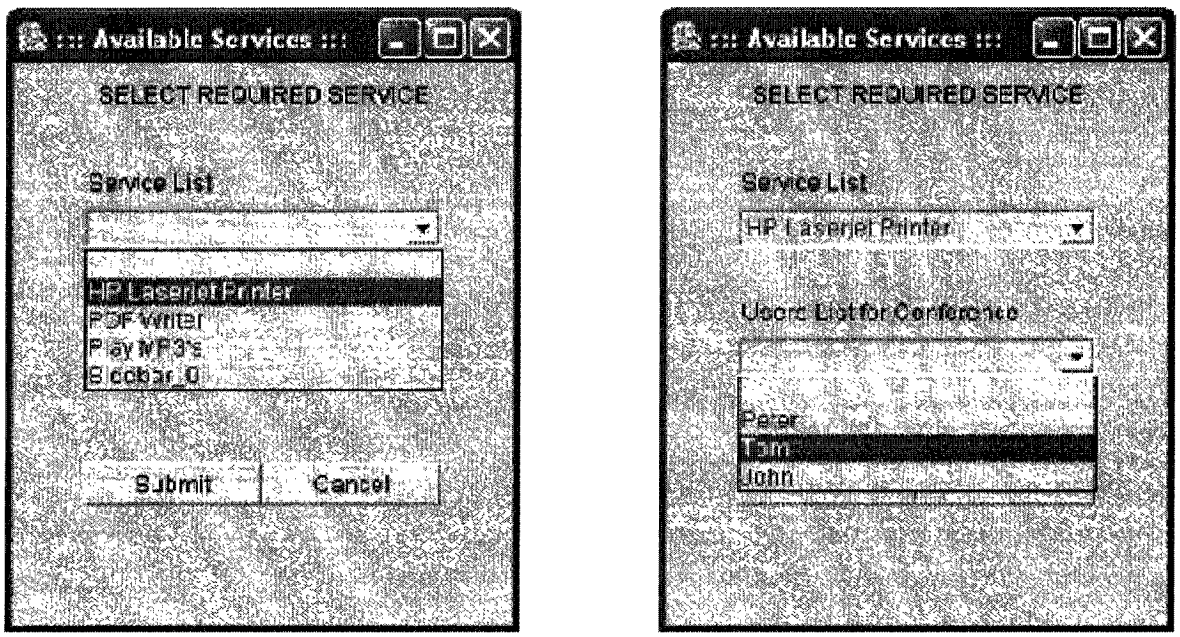

Figure 5.20: GUI displaying user and service lists in the conference

Sidebar_0 represents a conference session that is already active between some users and this new user may join that service if he wishes to. This user list gets automatically updated whenever there is a change in the presence of user or a service. The agent that maintains this list gets input from the CMS and reflects the change in all actively opened GUIs.

Figure 5.21 represents the class diagram of the prototype. Arrowed links denotes an inherited relationship while normal links denotes accessing of methods or variables by the other class.

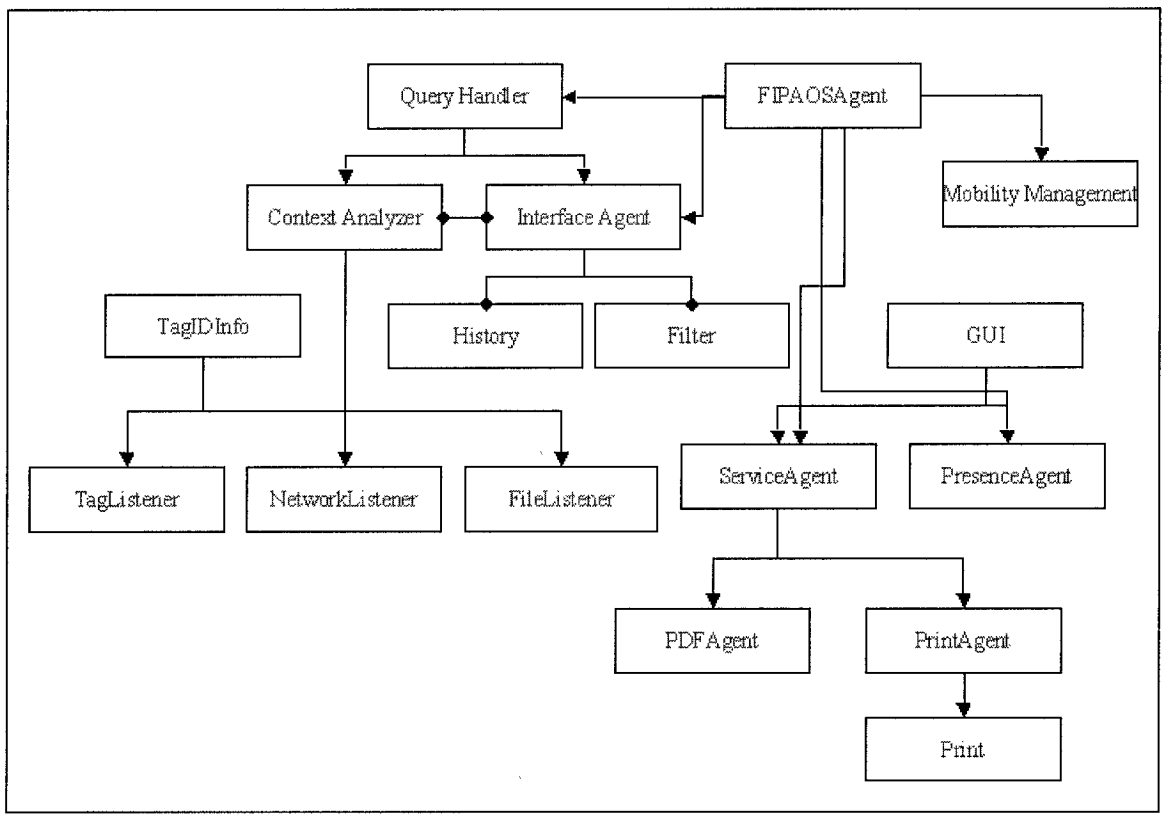

Figure 5.21: Class Diagram of the prototype.

Implementation Infrastructure

The CMS was implemented on a resource rich environment with wired and wireless LAN. The CMS server ran on a system with Pentium IV 2.4 Ghz processor with 512 MB RAM. The Printing Service agent was executed on a Pentium IV 1.4 Ghz processor with 1 GB RAM. The RF sensor to identify the physical presence of entities in a room is from RFCODE. The reader was equipped with 802.11 wireless protocol and connected to the wired LAN through a Orinoco Access Point. Two portable devices, a Compaq IPAQ 3800 Series running Microsoft Pocket PC and a laptop powered with Celeron 700 Mhz processor with 128 MB RAM running windows 2000 were used as terminal devices for mobile users. Both the mobile devices are featured with 802.11 wireless capability with the former using a LinkSys Wireless card and the latter using an Orinoco wireless card. The laptop was loaded with a PDF converter, which the user of the laptop can opt to share with the environment. In addition, four Pentium IV Windows XP desktops with 1.4 Ghz processors and 1 GB RAM connected to the wired LAN through 100 Mbps Ethernet card were used as terminals for users who do not possess a personal device.

5.3 Performance Evaluation

The performance of the prototype was measured by monitoring various parameters that influenced the prototype design. They are described in detail in the following passages. The infrastructure used is explained in the previous section.

Agent Deployment time

The most important parameter that was evaluated is the agent deployment time. This is the time difference between the entry of an entity and the deployment of agent, where the agent is ready to carry out requests. There are four steps in this process.

Detect time: The time taken to identify the entity once it has entered the room.

Object Creation time: Time taken to create context object.

Agent Creation Time: Time taken to create appropriate agent for the entity

Archival Time: Time taken to create or update an entry in the archive

The following figure illustrates the participation of each of these phases towards the total time taken for agent deployment.

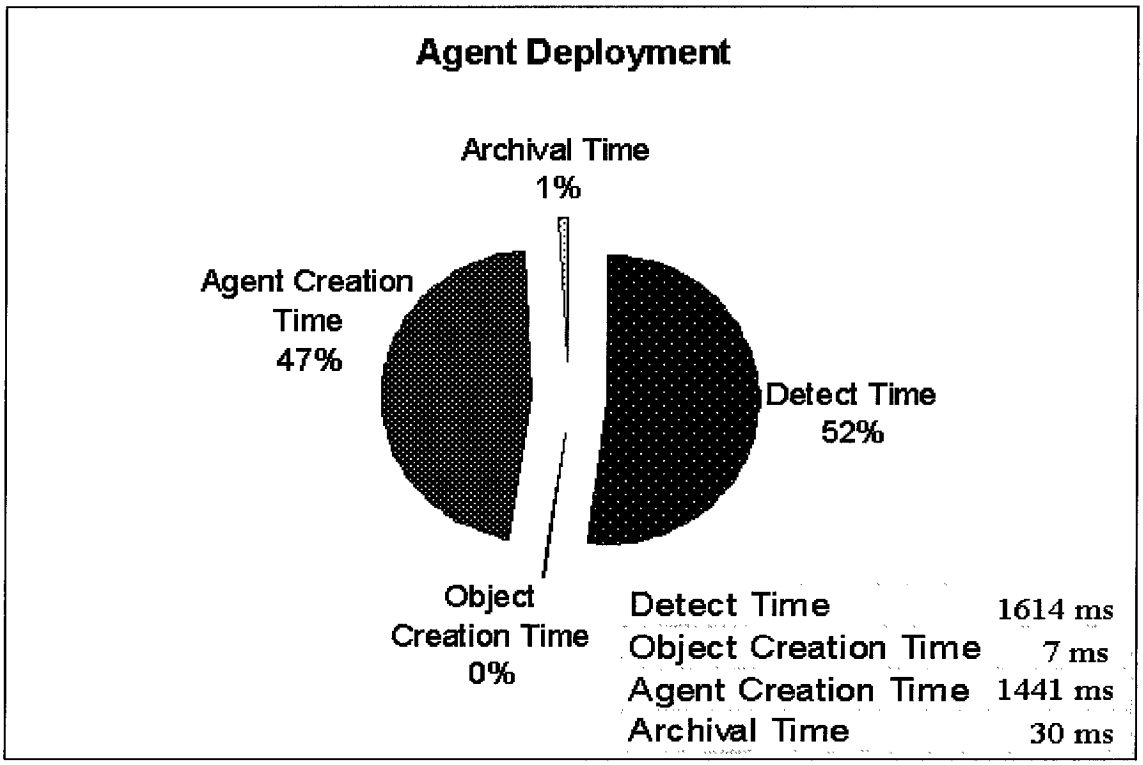

Figure 5.22: Total time taken for Agent Deployment

It can be noted from the pie chart that the detect time contributes the maximum percent of the total time. The main reason for this is the RF reader that streams presence data through sockets in a sequential manner and the profile access time that is required to identify the corresponding profile of the entity. When there is more than one entity entering at the same time, the presence information is obtained one after the other and thus there is a delay due to the queue. All these factors add up to form total detect time. Object creation time is almost 0% as it takes only 7ms. This indicates the efficiency of the storage block in the model. The main action that takes place in this step is the parsing of the profile and updating a few attributes in the profile.

Agent creation time is the time taken to identify the appropriate agent type (Service or Person), create the agent through FIPA-OS and then load the agent with the profile of the entity. This takes up considerable amount of time due to the time taken for the agent platform in creating a new agent. Archival time is also negligible since the time delay is mainly because of the creating a database connection with MS-ACCESS and when there is more number of entities accessing the database at the same time, this time value may increase.

Agent Deployment Versus Agent Shutdown

The comparison of the agent creation and shutdown time is discussed in this section. The creation time and shutdown time does not include the delay caused due to the RF reader in identifying the absence of the entity. The time charted in the following figure is the time from the agent receiving the notification about the presence of entity

from the reader till the shutting down of the agent. The time delay is caused mainly because of the agent platform and the archival operations performed by the agent.

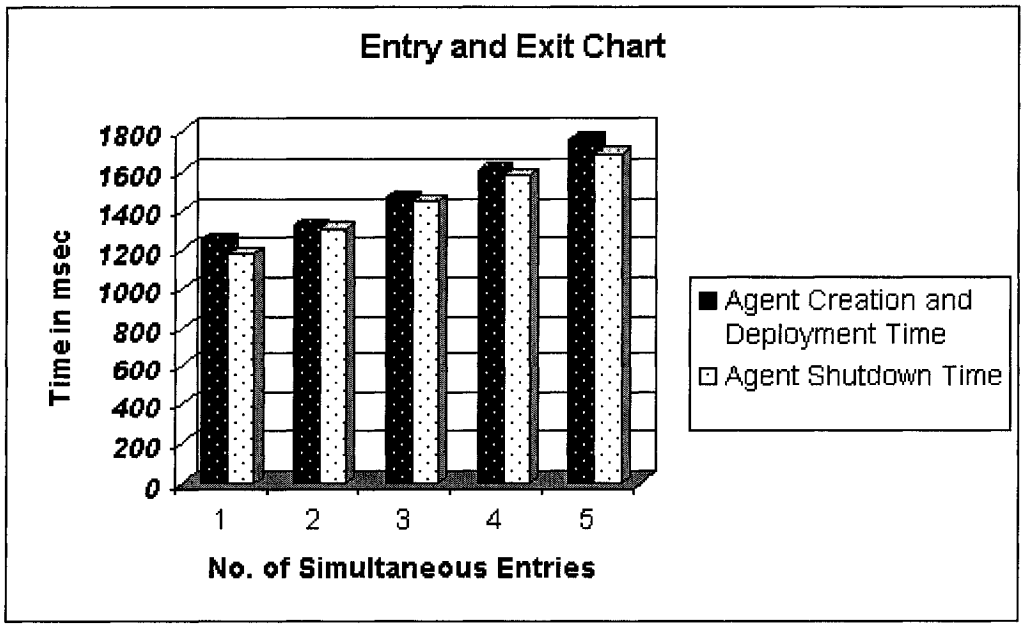

Figure 5.23: Graph comparing agent creation and shutdown time during the entry and exit of an entity

It can be noted that there is not a significant difference between creating and shutting down an agent as they perform more or less the same type of operation during creation and exit.

Memory Usage

Another important part of the performance evaluation is the amount of memory occupied by the entities. The main source of entity information comes from the XML profile. There are different versions of the same profile active for each entity during its course of stay. The average amount of total hard disk memory used by each entity is **8Kb.** When the entity is not present in the room, it occupies **5.5Kb** of memory to store its archived information. On the whole the prototype occupies **6.05mb** of hard-disk memory.

The size may increase with the number of entities in the room as the profile of those entities also adds to prototype memory.

Filtered Requests

The following chart shows the average number of requests that are filtered by the filter module. It proves the inevitability of the filter module in this prototype model.

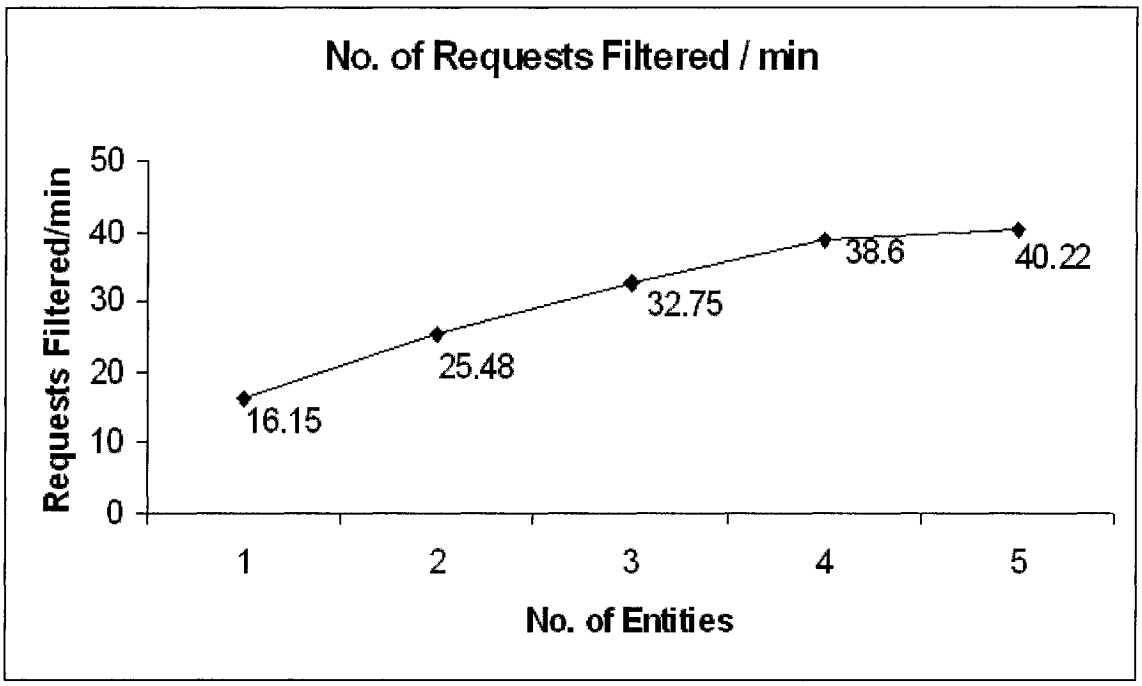

Figure 5.24: Average number of Filtered requests

The number of requests is due to the constant streaming of presence information of every entity by the reader. Whenever there is a beacon received from one of the tags, the CMS is informed. In addition, there is also beacon information from tags that are not associated with the room. The filter performs check on all these data and those requests that are rejected by the filter are plotted in the above graph. It can be observed that there is not significance increase in the number of beacons as the number of entities increases. This is because when the number of beacons in the room increases, the rate at which they

get lost before being read by the reader also increases. This loss becomes significant when there are relatively large numbers of tags in the room.

5.4 Feedbacks

During the process of implementation a good number of feedbacks and suggestions were received from Mr.Gray, Mr. Liscano, and Mr. Impey which helped significantly in the learning process.

The interesting one among them was the usage of some of the ubiquitous gadgets such as telephones as sensors to detect the availability of a person. Most of the discussions were targeted towards identifying the presence of a user and the automatic status identification rather than interrupting the user. For example, if only one user is identified in a room and the telephone is busy, the status of the user can automatically be changed to "On the Phone" until the telephone is available.

Another suggestion was to avoid using tags for hardware services and detect them using their network connectivity. But this method had its own disadvantages, as the exact location of the service inside the room cannot be resolved. Furthermore there were difficulties while extracting the context description from services through this method.

5.5 Personal Experience

From the prototype model, a great deal of invaluable experience was gained in creating applications for mobile ad hoc environments. Some of the complexities that were hidden during the design were unraveled and posed challenges during the implementation process.

One of the challenges that were faced during this process was identification and interaction with the service provided by the user. The agents should exclusively provide the properties and capabilities of the service. The service has to be coupled with its owner, i.e., creating a binding between the user and the service was another challenging task. One such case is where the user may leave the room and the service provided by the user may or may not leave the room. Modeling such a situation where the expiry of the service was intriguing provided us deeper knowledge in policy creations for service association and disassociation.

The implementation of various sensor interfaces proved to be less complicated task as the focus was mainly on RF sensors and custom designed software sensors, which did not require complex programming structures.

Data management and data security provided competencies during implementation. The maximal usage of agent technology was experienced in these areas. The agent platform, though it impaired some properties of ad hoc environment, provided valuable security features. For example every agent in the agent platform is required to register with the platform and had to follow some basic steps and standards. Agents that are created by a different agent platform will have conversational problems with the agents in the host platform.

While transferring agents to a hand-held device, the size of the agent has to be reduced which created complications in enabling certain features of agents. To overcome this deficiency a proxy service was used that provided those features to agents whenever a request is obtained. This helped in realizing the use of proxy in mobile environments.

A rather interesting observation made was the backup of context data. Instead of keeping them as context objects they were converted to XML files because XML format helped in transporting documents and organizing the back up with less extra effort.

5.6 Summary

This chapter featured the implementation of CMS. The various features and mechanisms of implementation of CMS were discussed. Snapshots of CMS during various phases were presented along with code snippets to describe CMS implementation. The various scenarios were detailed and how CMS handles those scenarios were analyzed. The various suggestions and feedbacks that were received during the implementation were also analyzed. The experience that was obtained during this design and implementation showed the success of this model. The performances of the various modules were evaluated and were sketched in graphical form.

Chapter 6

Conclusion

Context management thus forms the basis of any context-aware system and the performance and capabilities are directly influenced by the way in which context data is handled. As the final chapter of the thesis, it sums up the contributions made towards context aware computing and the possible future research directions that may entail this work. It also features some of the related work in the field of context aware computing and how CMS is different from them.

6.1 Summary

There have been quite numerous research work conducted in the field of context and context-awareness computing. Context has been used in various types of applications in solving several problems or providing additional features. In this proposal, context was the prime object of interest and the ideas revolved around devising an efficient system geared towards context management. The ubiquity of wireless LANs and seamless availability of Internet connections in the recent years has paved way for mobile computing. The demand for handheld devices is constantly growing along with the capabilities of those devices. Thus powering the resource rich mobile environments with context awareness provides all the ingredients for a challenging task for researchers.

A good context management system design provides way for an efficient and best use of context. The thesis explained an attempt towards providing an efficient context management system for the ad hoc and mobile environments. As it can be seen here, context itself encompasses a galore of meaning with respect to every specific application. Thus creating a system that would provide a context management solution for all types of applications is an impossible task.

When it comes to dynamic creating and accessing of context, the system must be capable of handling any type of data at any point of time. Thus the need for constantly available specialist programming modules proved inevitable. This is where the agents become notable and as it can be seen in the design, several agents that perform specific tasks are employed. Another reason for choosing agents was the ease of integration of context management system with the mobile ad hoc collaborative system. Since the

whole model was conceived as agent-based, several problems relating to negotiations and data exchange between the modules were solved during the integration.

Thus in this research work a novel architecture to manage and manipulate context was presented. This architecture showed the use of context in smart automation of tasks by managing context and triggering appropriate actions. Context data relations were used to describe objects through context data objects. This proved to be a simpler and easy representation structure when considering frequent modification in the data object. This model also showed how different types of heterogeneous devices can be plugged into this architecture with minimal effort once this device supports the agents conforming to the common agent platform standard. Finally, the use of agents greatly simplified the design and implementation of various tasks which otherwise would have been cumbersome. For example, if the designing were not through agents the migration of code and communication between other modules would have not been this easy.

6.2 Future Research Directions

The CMS discussed here paves way for numerous research directions that enable to fine-tune the model and enrich it with more capabilities.

Composite Context

The CMS deals mostly with preliminary context data with lesser interest on obtaining *composite context*. Composite context may be the value of an attribute obtained from two or more context values. For example,

If a user is accessing a fixed telephone in a particular room, then the exact position of the user in the room can be obtained and in the process the resource request can be sent to the closest one near the user.

Creating and using composite context provides a real challenge as it brings up numerous scenarios for a single case. It in turn increases the complexity of the context management system and provides challenges in creating the right context data structure. This is because, in order to obtain composite context, the CMS must be able to gather related context data alone without extensive searching procedures. The identification of composite context must be quick since the situation must be described before it expires, which in turn means that time forms a crucial part in performance. Composite context, when deployed can attract a lot of interest to the application making invisible computing a reality.

Context History

Decision-making is one of the valuable capabilities provided through context awareness. When analyzed with respect to an ad-hoc environment, decision-making concentrates more towards users' preferences and location. Predicting a user location through context-awareness is one challenging research area. Context history in CMS can be further extended to achieve this target with the input from complex context calculations and probability techniques. Prediction and triggering action share a cause-effect relationship in CMS. Whenever CMS predicts the value of a context attribute, triggering an action invariably follows it. As a result, CMS has to be further equipped to support other tools and protocols.

The list of context data archived can be extended to perform accurate predictions over a number of attributes. The expansion of context history may involve ramifications in storage structures and data transfer mechanisms, there by providing enough ingredients for a worthy research direction.

References

[1] G.Chen and D. Kotz., *A survey of context-aware mobile computing research* Technical Report TR2000-381, Computer Science Department, Dartmouth College, Hanover, New Hampshire, November 2000.

[2] Weiser, M. *Some Computer Science Issues in ubiquitous Computing*, Communications of the ACM, Back to the Real World, Special issue on Computer Augmented Environments, ACM Press, vol. 36, no. 7, 1993, pp. 75-84.

[3] Maes, Pattie, "Artificial Life Meets Entertainment: Life like Autonomous Agents," Communications of the ACM, vol. 38, issue. 11, 1995, pp 108-114.

[4] D.B. Lange and D.T. Chang., *IBM Aglets Workbench - Programming Mobile Agents in Java,* IBM Corp. White Paper, September 1996.

[5] Emorphia. "FIPA-OS", http://fipa-os.sourceforge.net/.

[6] OMG (1995), Common Facilities RFP3, Request for Proposal OMG TC Document 95-11-3, Nov. 1995, http://www.omg.org/.

[7] M.Breugst, I. Busse, S. Covaci, and T. Magedanz. Grasshopper - A Mobile Agent Platform for IN Based Service Environments. In Proceedings of lEEE IN Workshop, Bordeaux, France, May 1998, pp 279-290.

[8] S. Fischmeister, G. Vigna, and R. Kemmerer., "Evaluating the Security Of Three Java-Based Mobile Agent Systems", In Proc. of IEEE Mobile Agents 2001, Lecture Notes in Computer Sciences, Georgia, Atlanta, USA, December 2001, pages 31-41.

[9] A. Orso, M.J. Harrold, and G. Vigna, "MASSA: Mobile Agents Security through Static/Dynamic Analysis," in Proceedings of the First ICSE Workshop on Software Engineering and Mobility (WSEM 2001), Toronto, Ontario, Canada, April 2001.

[10] Jansen, W., Karygiannis, T. *Mobile Agent Security,* Technical Report, National Institute of Standards of Technology (NIST), Baltimore, 1999.

[11] Jukka Riekki, Jouni Huhtinen, Pekka Ala-Siuru, Petteri Alahuhta, Jouni Kaartinen, Juha Röning; "Genie of the Net, an Agent Platform for Managing Services on Behalf of the User", Computer Communications Journal, Special issue on Ubiquitous Computing, vol. 26, issue.11, July 2003, pp. 1188-1198.

[12] Sadeh, Norman, Chan, Enoch, Shmazaki, Yoshinori, and Van, Linh, "MyCampus: An Agent-Based Environment for Context-Aware Mobile Services," Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices, Bologna, 2002.

[13] David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste, "Project Aura: Towards Distraction-Free Pervasive Computing",IEEE Pervasive Computing, special issue on "Integrated Pervasive Computing Environments", Volume 1, Number 2, April-June 2002, pages 22-31.

[14] MIT. "Project Oxygen", http://oxygen.lcs.mit.edu/.

[15] Pascoe, Jason, "Adding generic contextual capabilities to wearable computers", in the Proceedings of the 2nd IEEE International Symposium on Wearable Computers (ISWC'98), Pittsburgh, PA, IEEE. October 19-20, 1998, pp. 92-99.

[16] Schilit, B.N., Adams, N.I. and Want, R., "Context-Aware Computing Applications", Proceedings of the Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, Santa Cruz, CA, 1994, pp. 85-90.

[17] Pascoe, J., Ryan, N.S. and Morse, D.R., "Issues in Developing Context-Aware Computing. Proceedings of the International Symposium on Handheld and Ubiquitous Computing, Karlsruhe, Germany, Springer-Verlag, Sept.1999, pp. 208-221.

[18] M. Khedr, A. Karmouch, R. Liscano, T. Gray, "Agent-based Context Aware Ad hoc Communication", IEEE/IFIP MATA'02, LCNS 2521, Barcelona, Spain, Oct 2002, pp 105-118.

[19] Huang, Q., *Supporting Context-aware Computing in Ad Hoc Mobile Environments*, Technical Report WUCS-02-36, Department of Computer Science and Engineering, Washington University, St. Louis, Missouri, 2002.

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

[21] Rekimoto, J. Matrix, "A Realtime Object Identification and Registration Method for Augmented Reality" in Proceedings of the Asia Pacific Computer Human Interaction Conference (APCHI'98), Kanagawa, Japan, July 1998.

[22] Andy Harter, Andy Hopper, Pete Steggles, Any Ward, and Paul Webster, "The anatomy of a context-aware application." in Proceedings of the 5thAnnual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 1999), ACM Press, Seattle, WA, August 1999, pp 59-68.

[23] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan., "The cricket location-support system" in Proceedings of MOBICOM 2000, ACM, ACM Press Boston, MA, August 2000, pp 32-43.

[24] Peter H. Dana, "Global positioning system overview", *GPS* http://www.colorado.edu/geography/gcraft/notes/gps/gps.html.

[25] Bennett, D. Clarke, J.B. Evans, A. Hopper, A. Jones, and D. Leask, "Piconet - embedded mobile networking, " IEEE Personal Communications, vol. 4, no. 5, October 1997, pp. 8-15.

[26] Kaplan, Elliott D. ed. *Understanding GPS: Principles and Applications*, Boston: Artech House Publishers. 1996.

[27] E. Miller, *An Introduction to RDF*, Online Computer Library Centre, Inc., Dublin, Ohio, May 1998. http://www.dlib.org/dlib/may98/miller/05miller.html.

[28] W3C. "Platform for Internet Content Selection", http://www.w3.org/PICS/.

[29] Nick Ryan, "ConteXtML: Exchanging Contextual Information between a Mobile Client and the FieldNote Server", *ConteXtML*, http://www.cs.ukc.ac.uk/research/infosys/mobicomp/fnc/ConteXtML.html.

[30] Lalana Kagal, Tim Finin and Anupam Joshi, "A Policy Language for A Pervasive Computing Environment", IEEE 4th International Workshop on Policies for Distributed Systems and Networks, Lake Como, Italy, June 4-6 2003, pp 63-76.

[31] N. Damianou, N. Dulay, E. Lupu, M. Sloman, "The Ponder Policy Specification Language" Proc. Workshop on Policies for Distributed Systems and Networks (Policy 2001), Bristol, UK, Springer Verlag LNCS 1995, pp 18-39.

[32] Marty Hurley, Nimsha Meta, Rich Simon, and Mary Ellen Zurko. *Authorization for distributed applications and groups*. Technical report, OSF Research Institute, Cambridge, 1996.

[33] Christian Bettstetter and Christoph Renner, "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol" in Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School, Twente, Netherlands, September13-15, 2000.

[34] M. W. Subbarao, *Ad Hoc Networking Critical Features and Performance Metrics* White Paper, October 1999.

[35] Elizabeth Royer and C-K Toh, "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks ", IEEE Personal Communications Magazine, April 1999, pp. 46-55.

[36] C.E. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced DistanceVector Routing (DSDV) for Mobile Computers," ACM SIGCOMM: Computer Communications Review, vol.24, issue 4, pp.234-244, October 1994.

[37] Murthy, S. and Garcia-Luna-Aceves, J. J.; "An Efficient Routing Protocol for Wireless Networks", ACM Mobile Networks and Applications Journal (MONET), Special Issue on Routing in Mobile Communication Networks, Vol.1, issue 2, October 1996, pp 183-197.

[38] C.E. Perkins and E.M. Royer, "Ad-hoc On-Demand Distance Vector Routing," Second IEEE Workshop on Mobile Computing Systems and Applications, February 25-26, New Orleans, Louisiana, USA, 1999, pp.90-100.

[39] D.B. Johnson and D.A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in Mobile Computing, ed T. Imielinski and H. Korth, Kluwer Academic Publishers, chapter 5, 1996, pp.153-181.

[40] L. Zhou and Z.J. Haas, "Securing Ad Hoc Networks" IEEE Network Magazine, vol,13, issue 6, November/December 1999.

[41] Shivanajay Marwaha, Chen Kong Tham, and Dipti Srinavasan. "Mobile Agents based Routing Protocol for Mobile Ad hoc Networks" In Proceedings of IEEE Global Telecommunications Conference (GLOBECOM'02), Taipei, Taiwan, November 17-21 2002.

[42] Romit RoyChouldhury, S. Bandyopadhyay and Krishna Paul, "A Distributed Mechanism for Topology Discovery in Ad hoc Wireless Networks using Mobile Agents", Proc. of the First Annual Workshop On Mobile Ad Hoc Networking & Computing (MOBIHOC 2000) in conjunction with IEEE/ACM Mobicom 2000, Boston, Massachusetts, USA, August 11, 2000.

[43] Indulska, J., et al. "Experiences in using CC/PP in context-aware systems," to be published in Proceedings MDM2003, Melbourne, 2003.

[44] J. Schirmer, H. Bach "Context-management within an agent-based approach for service assistance in the domain of consumer electronics", IMC 2000 Proceedings,Rostock-Warnemünde, Germany, November 9-10, 2000.

[45] Ajay D Kshemkalyani, George Samaras and Andrew Citron, "Context management and its applications to distributed transactions", Distributed Systems Engineering, Volume 5, Issue 1, pages 1-11.

[46] Paul Castro and Richard Muntz. *Managing context data for smart spaces*, IEEE Personal Communication, October 2000, pages 44-46

[47] Roman, G.-C., Julien, C., and Murphy, A. L., "A Declarative Approach to Agent-Centered Context-Aware Computing in Ad Hoc Wireless Environments," 1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'2002), co-located with ICSE'02, Orlando, FL, USA, May 2002, pp 94-109.

[48] A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

[49] Universal Plug and Play forum. "Universal Plug and Play Device Architecture", Version 1.0, http://www.upnp.org/download/UPnPDA10_20000613.htm

[50] Eric Guttman. Service Location Protocol: Automated Discovery of IP network Service. IEEE Internet Computing, vol. 3, issue 4, July 1999, pp 71-80.

[51] Sun. Technical White Paper. "Jini Technology Architectural Overview", http://wwws.sun.com/software/jini/whitepapers/architecture.html.

[52] Salutation Consortium. "Salutation Architecture Specification", 1999. Version 2.1 http://www.salutation.org.

[53] Bluetooth Special Interest Group. "Specification of the Bluetooth System 1.0b, Volume 1: Core", http://www.bluetooth.com, Dec. 1999.

[54] A. Ohsuga, Y. Nagai, Y. Irie, M. Hattori, and S. Honiden. Plangent: An Approach to Making Mobile Agents Intelligent." IEEE Internet Computing, vol. 1, issue 4, July 1997.

[55] Macker, J. and Corson, S.; "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations", RFC2501, January 1999, http://www.ietf.org/rfc/rfc2501.txt.

[56] X-10 (USA). "X10 Protocol", http://www.x10.com/.