NOTE TO USERS

This reproduction is the best copy available.

UMI®

Université d'Ottawa · University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DE ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Mohamed KHEDR

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

Ph.D. (Electrical Engineering)

GRADE - DEGREE

School of Information, Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

An infrastructure for Managing Context Information in Pervasive Computing Environments

A. Karmouch

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

B. Esfandiari

A. Nayak

S. Pierre

T. Yeap

J.-M. De Koninck, Ph.D

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

An Infrastructure for Managing Context Information in Pervasive Computing Environments

By

Mohamed Essam Khedr

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the degree of Doctor of Philosophy

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

University of Ottawa

Ottawa, Ontario, Canada

November 2004
Copyright © Mohamed Essam Khedr, 2004

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-494-01721-X
Our file Notre référence
ISBN: 0-494-01721-X

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

The increasing popularity of mobile devices, such as mobile phones and personal digital assistants, and advances in wireless networking technologies, are enabling new classes of applications that raise challenging problems to application developers. These applications have to be aware of the variations in the execution context such as location, time, user activities, and device capabilities in order to tune and adapt their intended functionalities.

We argue that developing and managing applications that are context-aware would be extremely hard, tedious, and error-prone if not supported by a computing infrastructure. This is because application developers would have to deal with issues such as context interpretation, context reasoning, and context adaptability, and consequently would be distracted from the actual requirements of the applications they are developing.

We investigate the principles of ontologies, negotiation, and approximate reasoning, and their usage to support context-awareness and dynamic adaptation for applications in pervasive environments. These principles are integrated in our proposed infrastructure, which offers application developers a set of APIs and computing components to facilitate the process of developing and managing context-aware applications.

The APIs that the infrastructure provides allow users to manipulate the information encoded in their semantic profiles and to negotiate context information causing the infrastructure to tailor its behaviour to applications' need. Accordingly, the infrastructure reasons and makes automated decisions that are based on this negotiated contextual information to achieve adaptability and to cope with the frequent changes in the environment.

To manifest the effectiveness of these principles in developing and managing context-aware applications, we discuss the architecture and implementation of an agent-based context-aware infrastructure that implements these principles, and report on performance and usability results obtained from a thorough evaluation of the infrastructure.

Acknowledgements

I would like to thank my supervisor, Professor Ahmed Karmouch for his support, his open-minded attitude, and his creativity.
Professor Karmouch always gave me freedom to choose my own path and helped me to get back on track when I got lost. I learned a great deal from him, especially the importance of a positive attitude to address new challenges. I am grateful to him.

I would also like to thank my colleagues at the Multimedia and Mobile Agent Lab, MMARL, and the good time I spent with them for the past four years.

I am indebted to my parents for their love and unconditioned support and for more than I will ever be able to express.

I am grateful to my family, Salma, Danah and Nada, for their understanding and the support I found to accomplish this thesis.

Dedication

To my mother, father and my lovely family for their unconditional support throughout all these years

Table of Contents

List of Figures and Illustrations

List of Tables

List of Symbols and Abbreviations

ACAI: Agent-based, Context-Aware Infrastructure

ACF: Agent Configuration File

ACL: Agent Communication Language

CIF: Context Information File

CLNP: Context Level Negotiation Protocol

CMA: Context Management Agent

CPA: Context Provider Agent

FOAF: Friend of a friend

FOL: First Order Logic

IA: Inference Agent

IDL: Interface Description Language

IFR: Inductive Fuzzy Reasoning

IFT: Inductive Fuzzy Tree

IMS: Instant Messaging System

JADE: Java Agent DEvelopment framework

MF: membership function

OA: Ontology Agent

OWL: Ontology Web Language

RDF: Resource Description Framework

RDQL: Resource Description Query Language

SA: Service Agent

SKBA: System Knowledgebase Agent

UA: User Agent

XML: eXtensible Markup Language

Chapter 1

Introduction

1.1 Overview

People today are getting more mobile, and more likely to be equipped with wireless devices whose hardware, software and computing capabilities are becoming evermore powerful [2],[3]. Pervasive computing [1],[4],[5] promises that as users move across different computing domains, they will remain transparently connected to the infrastructure in such a way that seamlessly enables them to continue gaining access to dynamic range of applications and services.

Unfortunately, current networking and computing infrastructures do not fully support such model of pervasive computing [6]. For example, if a mobile user today wants to use the computing resources of a new environment, he has to manually or through an administrator figure out how to continue his activities using the local resources of that new environment and in such a way that resembles his usual actions in his home environment.

This is unacceptable in pervasive computing environment since it will not smoothly scale with the increasing number of services, unpredicted user mobility, and frequent changes in the environment context.

1.2 Challenges in Pervasive Computing

Pervasive computing means embedding computing and communication technologies into resources and artifacts to ease the interaction between humans and computers. This raised issues that cannot be provided by the core components of traditional infrastructures. This is because pervasiveness destroyed the constancy assumption that is prevalent in traditional distributed systems [2],[7].

Infrastructures are now required to react to the dynamic nature of pervasive computing environment, such as frequent changes of location, limited capabilities of devices, variability of network resources, and so on. They need to face temporary and unpredicted loss of network connectivity, when their hosts are on the move. They need to discover other hosts and services in an ad-hoc manner since they are usually engaged in short connection sessions [8].

The problem escalates with current handheld devices. These devices are gaining more computational power and larger memory sizes. Today, a typical laptop includes a CPU that is as powerful as a desktop, and current PDAs have at least 64 MB of memory and equipped with latest wireless technologies such as IEEE 802.11 and Bluetooth. In addition, applications are being developed that bring the desktop environment to users' handheld devices such as instant messaging, SMS, email, internet browsing and more. However, these devices introduced the second contribution to the problem of building infrastructures for pervasive environments, i.e. interoperability.

Applications developed for these types of devices are proprietary in nature and cannot be extended or reused by other applications and developers. Moreover they are context insensitive. They cannot follow what is happening around them. Hence, they cannot take timely, context-centric decision.

Context raised the most significant complication in building infrastructures that support pervasive environments [9],[10],[11],[12]. This is accredited to the necessity to model vast types of contexts that are vague in nature, imperfect and interdependent. By context, we mean any information that characterizes the situation of an entity [13]. An entity is a

person, a place, or an object that is considered relevant to the interaction between users and applications. The definition points out that context-aware applications look at who, where, when and what information and use this information to determine why a situation is occurring.

Pervasive environment has vast types of context information. This includes physical contexts (like location, time), environmental contexts (like weather, light and sound levels), information contexts (like stock quotes, sports scores), personal contexts (like health, mood, schedule, activity), social contexts (like group activity, social relationships, other people in a room), application contexts (like email received, websites visited) and system contexts (like network traffic, status of printers) [14].

Context also breached the concept of transparency that was the key concept in the success of traditional infrastructures supporting distributed computing environments.

Transparency means that the infrastructure components hide the environment information, i.e. context, from application developers and unilaterally manage the seldom changes that might happen in the environment. While this has been a good design approach in traditional infrastructures, it did not succeed in context-aware pervasive environments.

The main reason for the failure of the transparency concept is that context spans the whole infrastructure layer architecture, from the physical layer up to the application layer. Context of each layer is not isolated from those of the other layers but are interdependent. Therefore, by hiding environment information from the application layer and the inability to adapt to the application's context has lead to the failure of traditional infrastructures. For example, a user might need a high multimedia resolution delivered to his desktop if he is in his office and low resolution if he is using his PDA. Thus, user location, user preferences, and device capabilities are context information that must be used by the infrastructure components in order to satisfy the requirements of the user.

As a result, infrastructures must provide a dynamic, flexible, and simple mechanism for negotiating contextual information with the applications to set context levels that they can monitor, provide, and adapt their behavior accordingly.

As a conclusion, new infrastructures are required for supporting pervasive environments in general and context-aware environments in specific. These infrastructures must provide new solutions for modeling context, solutions for automated reasoning about context, and solutions for manipulating context in order to facilitate the development and management of context-aware applications in pervasive environments.

1.3 Agent-Based Context-Aware Infrastructure

With these challenges in pervasive computing and the immense amount of available context information, developing and managing context-aware applications become difficult to realize and costly to implement. Thus, we propose an agent-based context-aware infrastructure (ACAI), to overcome these challenges [15]. In particular, ACAI has the following capabilities:

- Represent context and meta-context information using ontologies.
- Adapt infrastructure behaviors using context negotiation protocol.
- Reason about uncertain contexts using inductive fuzzy reasoning
- Provide APIs for developing and managing context-aware applications.

This dissertation presents the design, formalization, implementation, and evaluation of ACAI, an agent-based context-aware infrastructure that has the aim to facilitate the development of context-aware applications suited for pervasive computing environments. In particular, we exploit the principles of ontologies, inductive fuzzy reasoning, and agent technologies to achieve a context-tailored service provisioning and automated system adaptation to explicit and implicit contextual situations.

Using ontologies for representing both contexts and meta-context information give agents the ability to process, share, and reason about their situational activities. Specifically, OWL [16] as a knowledge representation language provides rich axioms and constructs that are suitable for modeling various types of contextual information. OWL is based on description logic theory and hence information modeled using OWL can be reasoned over using any logical inference engine.

Realizing ACAI's core components using agent technologies [17] have two purposes. First, agents have the ability to model the infrastructure interactions as behaviors and have the capabilities of solving some of the traditional infrastructure drawbacks such as mobility and automated task accomplishment. Second, agents can acquire knowledge and adapt to the changing situations in the environments through their ability to negotiate, to exchange information and to reason about the exchanged information.

Finally, our decision to use fuzzy reasoning in general and inductive fuzzy reasoning in specific was attributed to the difficulty in managing the vagueness and imperfectness of context information and the inability to deduce correctly higher-level context information from this uncertain context[18],[19].

IFR exploits the fact that context with a high-level of abstraction depends on other low-level contexts and it can be determined (or predicted) based on these low-level contexts. For example, in our context-aware conference system it is important to classify whether a user is the presenter of the coming lecture or he is an attendee to decide correctly the time and method of alerting him with his next activity. This decision-making involves inferring over context information that is vague and interdependent in nature such as time, location, user activities, device capabilities and so on. IFR in ACAI uses training examples provided by application developers to build its fuzzy inference tree that is used to deduce actions or new high-level context information. Rules are then extracted from the generated tree and used for making decisions, predicting actions, and reasoning about context.

1.4 Thesis Contribution

The goal of this thesis is to investigate new principles and design new models and algorithms that would facilitate the development and management of context-aware applications. The following is an overview of the main contributions of this thesis:

1. Modeling of context and meta-context information using ontologies.
2. Reasoning about context using inductive fuzzy reasoning.
3. Adapting infrastructure behaviour by negotiating context specifications.

1.4.1 Semantic Modeling of Context and Meta-Context Information

We have investigated the use of ontologies as the technique for semantically modeling context information, negotiation process, and fuzzy reasoning mechanism. The aim was to provide a common understanding of the different contexts and interaction mechanisms used by ACAI's core components which will allow application developers to invoke, exchange and manipulate context information in a unified and extensible manner [20],[21],[15].

Ontologies define a common vocabulary for entities that need to share information about a certain domain. They are machine-interpretable definitions of the basic concepts in the domain of interest and the relations among them. Moreover, ontologies provide the ability to reason and deduce implicit information from the constructs defined in the ontology, enable the reuse of domain knowledge, make domain assumptions explicit and separate domain knowledge from operational knowledge.

We used the emerging semantic web standard, OWL, to represent ontologically data and system information of ACAI. Data in ACAI is the context information while the system information is the negotiation process and the fuzzy terminologies.

The negotiation performatives, involved actors, and utility functions are examples of the information modeled in the negotiation process while membership functions, hedges, and degree of similarity are examples of the terminologies modeled in the fuzzy ontology.

Our novel approach of using ontologies to model the meta-information of the infrastructure enables the infrastructure to understand what it performs as a service, how to adapt implicitly to changes in the environment, and how to interact with other entities in the environment.

Ontologies representing the contextual information of entities in the environment are formulated in the form of profiles. Profiling is the way of representing the extent to which something exhibits various characteristics. This includes the context preferences of devices and services, the context preferences of users, and the context preferences of sessions managed by the infrastructure. Profiling has the advantage of achieving the separation of concern, i.e. what the infrastructure requires as data input is separated from how it uses these data.

The semantic modeling of context and meta-context information paved the way for our second contribution, the adaptation of ACAI's behavior based on context.

1.4.2 Adaptability Using Negotiation

Dynamic adaptation to the vast quantity and diverse quality of context information, in addition to the fact that no user is interested in all available contexts, led us to the concept of context negotiation. This means entities in the environment negotiate context specifications using the context ontologies [22],[23] ,[24],[25].

The negotiation process tends to set the context information, in terms of parameters, conditions and actions, which are required by users or applications from one side and can be provided by the context providers from the other side. The negotiation is based on a multi-attribute utility function for decision-making and stopping criteria. The outcome of the negotiation will enable the infrastructure to perform the dynamic association between entities and contexts and to encode such association using ontologies.

This concept has the following unique advantages:

- A declarative method with which users state their context requirements and context providers offer dynamic customized context-based service information.

- Protection against unexpected performance degradation because context providers know users' need and can thus better manage their resources.
- Allow seamless context-based provisioning of services across domains in large environment.
- An easy mechanism with which context providers can intelligently supply and monitor contextual resources.

1.4.3 Reasoning About Context Using Inductive Fuzzy Reasoning

User needs may vary over time, application requirements may differ from one situation to another and service capabilities may not be appropriate from on session to another. Thus, ACAI should be able to infer, in an autonomous manner, the actions and decisions to take and deduce implicit contextual information [26],[15].

Induction in general is the process of reasoning from a given set of facts to arrive at general principles or rules. For example, if I like football, baseball, and basketball, a person might correctly conclude by induction that I like sports in general. Induction looks for patterns in available context information to infer reasonable conclusions. Induction is also viewed as a search through the problem space for a solution to a problem. The problem space is composed of the problem's major concepts, linked together by an inductive process that uses examples of the problem.

Using induction in ACAI to support context-aware applications has the following advantages:

- Discover rules and decide actions from captured and stored contexts.
- Induce higher-level contexts by uncovering patterns in captured contexts.
- Eliminate irrelevant context factors used in deciding actions and triggering events.

However, standard induction has the problem of sensitivity. It is highly sensitive to noisy input information as well as missing or corrupted values. Unfortunately, this is the usual case in context-aware environments as context is vague and imprecise. Fortunately, our decision to model context using ontologies provided us with a solution: the inductive

fuzzy reasoning (IFR). This is because fuzzy logic uses linguistic terms and linguistic variables, which are easily mapped to the semantically modeled context. IFR would enhance the inference capability of ACAI with its ability to cope with uncertainty and vagueness in captured context.

1.4.4 System Design and Implementation

To prove the effectiveness of the principles we have investigated in developing an infrastructure for pervasive computing environments, we have designed and implemented a multi-agent system, which realizes the infrastructure architecture and its components.

According to the functionalities that the application needs, this multi-agent system can be easily extended and deployed in possibly any context-aware environments (e.g., new agents can be created and registered, new context can be represented, and new services can be added).

The developed infrastructure provides a set of APIs for gaining access to the components of the infrastructure. This set of APIs follows the standard methods of software development such as modularization, separation of functionalities and the provisioning of public accessibility methods.

We designed and formulated the different profiles used to represent context and system information using ontologies and we developed the API required to manipulate and manage these profiles.

Finally, we implemented some running examples and conducted extensive experiments to evaluate the infrastructure performance and its applicability to support context-aware applications in pervasive environments.

1.5 Thesis Assumptions

Developing an infrastructure for pervasive environments in general and context-aware in specific requires a collaboration among researchers working in different aspects of research such as networking, communications, service discovery, knowledge representation, event handling, fault tolerance and more. This thesis will tackle parts of these aspects while making reasonable assumptions about the rest of them.

First, we assume the existence of a communication and networking layer that provides the required connectivity and communication mechanisms in both wire and wireless domains. We assume that this layer is available in our infrastructure and assists in providing the required contextual information, (such as location, identity and network characteristics), to the developed applications. It also provides the required medium for the different components of the infrastructure to interact.

Second, we assume that the infrastructure has the ability to discover and advertise the availability of services and users in the environments. This does not contradict with our service discovery application as it uses this assumed low-level discovery mechanism to refine the process of discovery and provide services according to the modeled and inferred contexts.

Third, we assume that the infrastructure is equipped with the security features needed to preserve users' privacy and prevent most of the security-threatening issues.

Finally, we assume the existence of vast types of sensors and other types of context providers that capture and provide row context information to the higher-layer of the infrastructure. This raw information is then modeled using our ontologies and managed and manipulated by the other components of the infrastructure.

1.6 Thesis Outline

The thesis is organized as follows. In Chapter 2, we discuss work related to this thesis and provide a critical overview between these works and our proposed infrastructure. We also present a motivating scenario that highlights the importance of context in pervasive environments, and we use it throughout the thesis as our working example. Chapter 3 presents ACAI architecture. This includes the multi-agent system that constitutes the different functional components of ACAI, their responsibilities and the adaptation capability implemented in the form of context negotiation. In Chapter 4, we describe the semantic modeling of contexts and meta-context information using ontologies. We also present the extensibility and reusability mechanisms supported in these ontologies. In Chapter 5, we explain the reasoning capabilities of ACAI and present the algorithms and heuristics used to implement these concepts. Chapter 6 presents the detailed implementation of ACAI, the difficulties we faced in the implementation, and the methods we followed to solve these difficulties. This is followed by the experiments we performed to evaluate ACAI quantitatively in terms of performance (example time to reach an agreement in the negotiation, inference response time), and qualitatively in terms of ACAI effectiveness in developing context-aware applications. A critical review of the results concludes the chapter. Finally, Chapter 7 summaries the contributions, lessons learned from developing ACAI and future work.

Chapter 2

Background and Related Work

This chapter presents an overview of previous and current research projects in the fields related to my PhD thesis, particularly context awareness, pervasive computing and semantic knowledge representation.

I will focus on relevant projects, their use of context, the main features characterizing these projects and the differences between them and the work presented in this thesis. The chapter concludes with a motivating scenario to clarify the scope and perspective of the thesis and at the same time as a working example used throughout the thesis to exemplify the concepts introduced in this research.

2.1 Context Awareness

Context-aware computing is a new paradigm with which the system supports richer interaction among its entities based on their contexts (such as users location, time of activities, nearby people and devices, and devices preferences) [9],[27],[60],[61],[62]. Since it was proposed about a decade ago, with the genuine vision of Wiser [1],[2], many researchers have studied this topic and built several context-aware applications to demonstrate the usefulness of this new paradigm.

As introduced in Chapter 1, context-aware applications and the infrastructure that supports these applications are extremely difficult to build and maintain. This is due to lack of standard and efficient methodologies for developers to follow in order to build successful context-aware applications, and due to the diverse nature of context and

sources that capture context. I will present in the following section the theory and background about context-aware computing followed by related work in this field.

2.1.1 Context-Aware Theory

Context, as a concept, emerged from a group of interrelated areas of research. It started from the vision of mobile computing, through ubiquitous and pervasive computing, and reaching ambient computing. In each of these fields, context was the conclusion that researchers recommended and explored to enhance human-computer interaction and provide the seamless computing concept, i.e. "all the time, everywhere" .

Turning to context, researchers faced their first challenge, which was "what is context" and "what defines a context". No exact definition of context was agreed on until the time we were writing this thesis. Consequently, context was often misunderstood and in other cases, it opened the door for imagination and innovation in different fields of computing.

Schmidt, et al. [12],[63],[64],[65] described context as "*Context is knowledge about the user's and device's state, including surroundings, situation, and to a lesser extent, location*". They stated the important aspects of context to be where you are, whom you are with, and what resources are nearby, as well as the constantly changing execution environment [66]:

Dey and Abowd [11],[13] defined context as "*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*".

The definition of Dey is more general in explaining what is context than Schmidt's definition, as it relates context to any information and the relevancy of this information. However, these definitions lack one essential component; they do not cover the aspect of the context state itself. In other words, these definitions are describing the external behaviour of the surroundings in which context is used and not the characteristics of context. These characteristics of context should define the context domain, its range, and its functionality independent of the surroundings where it is used.

Thus, we modified Dey's definition of context as follows:

"Any information that can be characterized and characterizes the situation of an entity"

Context characterization is the ability to define context values from the information universe of discourse and to define its applicability from its associated domain.

2.1.2 Related Work

2.1.2.1 Microsoft EasyLiving

The EasyLiving project focuses on supporting context-awareness in home and work environments, and states that computing must be as natural as light [70],[71],[72]. The main features of an EasyLiving environment are self-awareness, seamless access, and system extensibility.

The system uses computer vision to recognize gestures and users, to detect the location of users and to allow interfaces to move according to the location of the users. The system also uses this information to customize the services in the environment.

A major focus in the EasyLiving project was to model the geometry of the intelligent environment as accurate as possible in order to inform applications about the physical relationships between people and devices. Applications developed as part of EasyLiving project include dynamic remote control that provides access to services that are co-located with the user, teleporting of displays to the user's current location, personalizing media devices to play audio or video according to the interests of the user [73].

The architecture developed for EasyLiving uses middlewares and individual software components to provide support for separation of concerns, transparent communications, and constant availability of components to multiple applications. A middleware called InConcert has been developed to support communications between distributed components, using asynchronous messages constructed in XML format.

EasyLiving has partial support for specifying what context a component or application is interested in and for resource discovery. There is no explicit support for reasoning about context or context manipulation.

2.1.2.2 HP CoolTown

CoolTown [74] is an infrastructure that supports context-aware applications by representing each real world object (including people, places, and devices) as a Web page [75]. Each Web page dynamically updates itself as it gathers new information about the entity that it represents.

CoolTown primary aim is supporting applications that display context and services to end-users. For example, as users move throughout an environment, they will see a list of available services for this environment. They can request additional information or can execute one or more of these services.

CoolTown major work was extending web technologies, wireless networks, and portable devices to create a virtual bridge among mobile users, physical entities, and electronic services.

The CoolTown infrastructure [76],[77] provides abstraction components (URLs for sensed information and Web pages for entities) and a discovery mechanism to make the building of these types of applications easier. However, CoolTown was not designed to support infrastructure adaptability based on applications', and users' context. In addition, while CoolTown focuses on displaying context, it was not intended to support other context-aware features such as executing services based on context or reasoning about context to provide higher levels of contextual information.

2.1.2.3 Context Toolkit

The Context Toolkit aims at simplifying the development and deployment of context-aware applications [78],[79],[80]. It consists of three main abstractions:

- **Context Widgets.** Just as GUI widgets mediate between a user and an application, context widgets mediate between the applications and their operating environment. Every context widget encapsulates information about a single type of context, such as location or activity. They provide a uniform interface to components or applications that use context while hiding the details of the underlying context-sensing mechanisms. Context widgets support traditional pull and subscription mechanisms.

- **Context Aggregators**. Context aggregators can be thought of as meta-widgets. They take all the capabilities of widgets and in addition, they provide the ability to aggregate context information of real world entities such as users or places. Aggregators acquire this context from existing widgets thereby providing an additional degree of separation between the acquisitions of context from the use of context. By acting as a gateway between applications and elementary widgets, aggregators hide even more complexity about the context-sensing mechanisms.
- **Context Interpreters**. A context interpreter is used in the context toolkit to abstract or interpret low-level context information into higher-level information. For example, they use low-level contexts such as identity, location, and sound level information in the environment to interpret high-level context such as if a meeting is taking place.

The context toolkit makes the distributed nature of context acquisition transparent to the applications. Applications do not need to know whether these context-aware components are executed remotely or locally. All of the components share a common communications mechanism (XML over HTTP). These components run independently of any single application, allowing them to be used by multiple applications [81],[82].

2.2 Pervasive Computing

This section provides a review on pervasive computing [4] in terms of its theoretical background and related work that deals with this topic. I will present major research projects that developed infrastructures and middlewares for pervasive environments and had context as the active component in their architecture. I will also compare and highlight the similarities and differences between my work and these projects in the last section of this chapter.

2.2.1 Overview of Pervasive Computing

Pervasive computing [83],[84] is the infrastructure or middleware needed to provide proactively a rich set of computing capabilities and services to users everywhere and all the time in a transparent, integrated, and convenient way.

Pervasive computing focuses on integrating computing devices with the physical environment. It extends mobile computing paradigm to include all kinds of devices (wire-line and wireless) that spontaneously interact with one another in repose to the mobility and diverse activities of their users. These interactions occur in a dynamically and frequently changing environment. Pervasive computing is therefore the umbrella for a group of research fields concerning device technology, networking, middleware, and applications.

Device technology addresses the computation capabilities embedded in devices such as sensors, active badge systems, wall-mounted displays, cell phones, PDAs, and the development of frameworks that facilitate interactions among these types of devices. Networking covers the development of protocols for communication and information exchange among devices in a point-to-point, multicasting or broadcasting method.

Middleware is the interface between the networking technologies and applications running on pervasive devices. The middleware is responsible for mediating all interactions with the network on behalf of the applications, managing the execution of applications using environment resources, and providing methods for media access and information manipulation.

Context is the active component in pervasive computing environments, since context is the perceptual information about the status of entities in the pervasive environment [67]. Thus, current research is oriented towards exploring and integrating context in infrastructures that supports pervasive computing.

2.2.2 Related Work

2.2.2.1 Gaia Middleware

Gaia is a middleware that manages the resources and services present in a physical space and provides an application framework to assist developers in building context-aware applications [86],[87],[88],[89]. It exports an application programming interface that hides the dynamism and complexity of the space, and allows developers to program the execution environment associated with the space. Gaia is built as a distributed system with three major building blocks: the Gaia Kernel, the Gaia Application Framework, and the Applications.

The Gaia Kernel contains a management and deployment system for distributed objects and an interrelated set of basic services that are used by all applications. The Component Management Core in Gaia kernel dynamically loads, unloads, transfers, creates, and destroys all the components and applications of Gaia. Current implementation of Gaia uses CORBA [85] as the communication interface to support remote interactions.

Gaia has five basic services: the Event Manager Service, Presence Service, Context Service, Space Repository Service, and Context File System [90]. Gaia applications use these services along with the Application Framework, to support applications that execute within an active space.

The framework provides application mobility, adaptation, and dynamic binding. Gaia's event manager service is responsible for event distribution in the active space and implements a decoupled communication model based on suppliers, consumers, and channels. Each channel has one or more suppliers that provide information to the channel and one or more consumers that receive the information.

Gaia applications may use context information to adapt their behaviour and accommodate to users situations and activities. The context infrastructure consists of a number of components, called context providers that provide information about the current context. This includes sensors that track the location of people, conditions within a room (e.g. temperature and sound) and other external conditions, such as weather and current stock prices. Gaia application framework provides mechanisms to construct, run and adapt existing applications to the active space context. The framework is composed of a

distributed component-based infrastructure, a mapping mechanism, and a group of policies to customize different aspects of the applications.

The application framework infrastructure is based on the Model–View–Controller pattern and it introduces new functionality to export and manipulate the bindings of the application components.

2.2.2.2 The Solar System

The Solar system consists of a centralized component called Star that processes subscription requests from applications and deploys operators to the appropriate *Planets* as necessary [91],[92],[93]. A Planet is an execution platform for Solar sources and operators, and it is responsible for tracking subscriptions and delivering events in the operator graph.

The Star maintains a representation of the operator graph and services' requests for new subscriptions. When the Star receives a new subscription-tree description, it parses the description, and matches the subscription tree against its internal operator graph to see if there are operators that can be shared and re-used. If the Star decides to deploy a new operator, it instantiates the operator object on one of the Planets that periodically register themselves with it. The Star determines which Planet should host the new operator by considering the Planet's load and network traffic between Planets. The Star maps the operator graph onto the planetary network to distribute load and avoid congestion.

When deploying new subscriptions, the Star informs the appropriate Planets to arrange a subscription-tree from their operators. Thus, every Planet maintains all the subscriptions of each of its resident operators. When an operator publishes an event, the hosting Planet delivers this event to all its subscribing operators and applications.

Applications running outside the Solar system use a small Solar library to interface with the Solar system. This library allows the applications to send requests to the Star, manage their subscriptions, and receive Solar events over standard network protocols.

Solar represents contextual information as events [94]. Sensors are referred to as sources and they publish a stream of events. An application subscribes to event streams. An event

stream is a sequence of events that are important to the application execution and adaptation.

2.2.2.3 The Reconfigurable Context Sensitive Middleware

The Reconfigurable Context Sensitive Middleware RCSM is a context-sensitive middleware, which provides a development and run-time support for applications that need context-awareness capability [95],[96],[97].

RCSM provides the adaptive object container (ADC) for runtime context acquisition. The RCSM interface definition language (IDL) compiler tailors ADCs to particular context-sensitive objects. The context-sensitive interface lists the contexts that the applications use, the actions they provide, and a mapping between the contexts and the actions that clearly indicates the values at which an action should be completed.

RCSM's runtime execution and communication support allow applications in multiple devices to adapt and manage their interactions with other applications due to changing contexts.

RCSM facilitates the development of applications and software components that need to be context-sensitive by representing them as context-sensitive objects. Then using the context-sensitive interfaces, RCSM determines which type of context to monitor and which method of the objects to activate in the current context.

RCSM's Context-Aware Interface Definition Language (CA-IDL) [98], which is based on CORBA's Interface Definition Language (IDL), addresses the extensibility of the middleware and the developed application objects by separating their interfaces from their implementations [99].

2.3 Semantic Knowledge

The last research field that I will present addresses the methods of representing and manipulating semantic knowledge [100].

2.3.1 Theory

Computing devices are evolving from being single, isolated devices to gateways and recently to a worldwide network of information exchange, business transactions, and value-added services that are personally customized to users' requirements. Hence, support for semantic information exchange and knowledge representation is becoming the key issue in current web technologies. Knowledge representation is thus a vital component for developing and managing applications in context-aware environments. This is highly contributed to the fact that context-aware environments are directly affected by their ability to qualify and interpret context.

Just as there is no single theory to explain human-knowledge organization or a best technique for structuring data in a computing system, no single knowledge representation structure is ideal. A designer engineer must choose the knowledge representation technique that best suits the expected applications' data structure. To accomplish this, different knowledge representation types are developed such as [101]:

- Procedural knowledge that describes how problems are solved and directions on how to perform these solutions.
- Declarative knowledge that describes what is known about a problem.
- Meta-knowledge that describes knowledge about knowledge.
- Structure knowledge that describes knowledge structure, concepts, and objects.

Several effective techniques for representing these types of knowledge are developed such as Object-attribute-value, rules, semantic networks, logic- and frame-based techniques [101],[102].

We will present two research projects that took knowledge representation as a crucial component in their infrastructure design and we will compare them along with the other projects against our system in the last section of this chapter.

2.3.2 Related Work

2.3.2.1 Adaptive Personal Mobile Communication

Theo in his dissertation presented a mobile interactive space as a guiding vision for developing a user-centric computing and communication infrastructure [103],[104],[105]. This infrastructure allows users to conduct personal activities and ensures that a network that connects them and all other things in the environment exists. The network offers uniform connectivity, naming and localization, and (wireless) mobility support between entities. Residing in these communication spaces are users, virtual objects, and artefacts that can communicate, move, and interact via representations of themselves as mobile agents.

Three key components are proposed in this project: an eXtensible Service Protocol (XSP), a Mobile Service Knowledge (MSK) representation, and an Active Context Memory (ACM) for enabling entities to maintain awareness [106].

The eXtensible Service Protocol enables event-driven negotiation of actions between peers based on context. XSP uses SIP event-mechanism for the discovery of other entities and registration of mutual service relations. While registering a service relation, entities use XSP events to exchange service knowledge about other entities' context. Once they have established relations, entities can use this knowledge to send events to other entities and invoke services. Quoted from Theo's thesis "XSP is a set of capabilities merged into one protocol. These capabilities are service discovery, event handling, object access, and negotiation".

MSK is a knowledge-base repository based on N3 notations[107], a short form of Resource Description Format (RDF) and adheres to the logical notations of the object-oriented programming in SICStus prolog. An XML representation of the Mobile Service Knowledge (MSK) can be obtained via XSP objects.

The ACM's data model consists of MSK objects representing perceived entities, relations between the data model's entity and other objects, and the behaviour attached to these relations (corresponding to events to which the entity needs to respond).

2.3.2.2 The Context Fabric

The Context Fabric argues that a set of core services in the form of an infrastructure coupled with a specification language for stating context needs can greatly reduce the effort in developing robust context-aware applications [108],[109],[110].

In particular, the context fabric proposes four core services – a Context Event service, a Context Query service, an Automatic Path Creation service, and a Sensor Management service – as well as a Context Specification Language for stating context information at a level above the programming language level. Combined, this will provide a high-level, flexible, and generally applicable model for subscribing to context events as well as querying for context state.

The context specification language allows application developers to state context needs at a high-level of abstraction. The Context Event is a service that allows applications to specify what context events they are interested in, using the Context Specification Language. The Context Event service then monitors these events and asynchronously notifies subscribed applications when those events occur. The Context Query is a service that applications use to query context state. The Automatic Path Creation Service is a service that takes context information as input and assembles the dataflow of sensors and software components to refine, fuse, and interpret the low-level sensor data to high-level context information. The Sensor Management service discovers and manages all local sensors used by other services in the context fabric infrastructure.

The goal of the Context Specification Language is to provide a flexible way of stating context needs [111]. Currently, it is based on XML language and supports subscribing to context events and querying for context state. Subscription requests have two parts. The first part specifies information about the subscriber, such as the method to send an event to the subscriber, the name of the event to send to the subscriber, and any existing sensor data and context data relevant to the situation. The second part describes the context event that is required by the application.

2.4 Comparison between ACAI and Current Research Projects

In the previous sections, I presented major projects in the fields related to my work. These fields are context awareness, knowledge representation, and pervasive computing. In general, I found the following observations that Figure 2-1 depicts.

- No other project explored these fields as an integrated approach to provide the essential components for developing and managing context-aware environments. Particularly, in situations where developers have no prior knowledge of the services in the environment and the functionalities they provide.
- Most of the presented projects used static and hardwired approaches to represent context and context-based situations. As a result, context-aware applications remain prohibitively expensive to build and maintain.
- Most of these projects lack effective and rich knowledge representation methods to describe entity's contexts and enable automated manipulation of these contexts.
- Most of these projects provide reasoning about context based on standard logic. Although a powerful tool for reasoning, we argue that uncertainty in context will obstruct the powerfulness of logical reasoning.

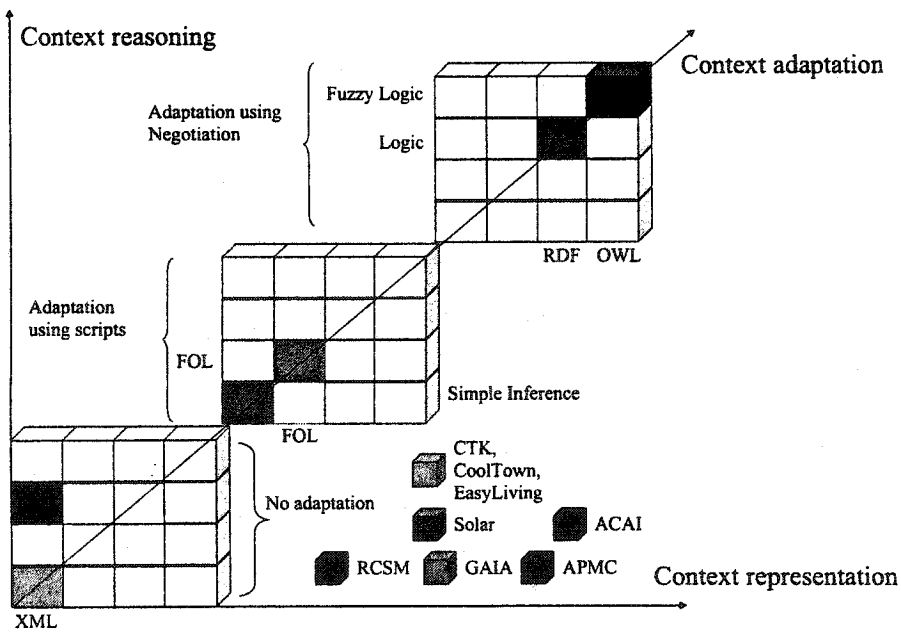


Figure 2-1 Comparison between related projects based on points of contribution

The Gaia project defines a metaoperating system for managing pervasive computing environments and physical spaces. Gaia recognizes the need for context information in its context service component. The context service component allows applications to query and register for any particular context information that helps them adapt to their environments.

A registry maintains a list of available context providers. Applications can use this registry to find the providers of the context they desire. Gaia uses first order logic for modeling context, which allows developers to write rules that reason about context information.

However, Gaia lacks the interoperability among different active spaces. One reason for this drawback is that Gaia is not an ontology-aware system. It uses proprietary scripts for system configuration and context representation. This ad hoc representation of context limits applications using Gaia to effectively share and exchange information about contexts of interest to them. Applications cannot reflect their context requirements to Gaia's kernel in order to allow Gaia to adapt to the applications' situation. Applications can only use any of the context providers that are available in the registry, independent of the context providers' point of view leading to rapid degradation in Gaia performance. This introduced scalability issues when the number of application instances that access the registry and interact with the context providers increases.

Gaia uses first order logic for reasoning about context while ACAI supports fuzzy reasoning, which gives ACAI an advantage over Gaia in vague and uncertain situations. At the same time, ACAI has similar reasoning ability in deterministic situations since fuzzy logic is the generalized form of standard logics.

On the other side, Gaia supports security and multiple-resource utilization per application, which ACAI currently does not support.

RCSM is based on ORB and provides what is called the adaptive object container (ADC) for context data acquisition at runtime. The RCSM-IDL compiler generates a custom-made ADC tailored for a particular context sensitive object. The context sensitive interface lists context information that applications use, actions that applications provide and mapping between the specified contexts and these actions that clearly indicates when

an action should be completed based on specific context values. ACAI outperforms RCSM in two situations. First, the capability of ACAI to separate context knowledge from context processing gives it the ability to work in situations where context is incomplete or ambiguous. Second, RCSM has no clear method for composing or reasoning with context. RCSM has to be re-programmed offline as new context sources are added. In addition, RCSM has no verification mechanism for validating context attributes. Application developer has to state the different situational actions and mapping between these actions and the context list at design time as RCSM has no methods for adaptation at runtime. The RCSM-IDL proposed by RCSM for representing context has its ancestor drawbacks. Users generally dislike the language mappings of IDL. In some cases, they feel the language mapping itself is unnecessarily complicated as the case of C++ mapping. In other cases, such as Java, some feel they should be able to use Java directly as their IDL, instead of being forced to use CORBA's mapping [122].

RCSM has the advantage of providing a context sensitive communication module, which ACAI currently does not support. This allows RCSM to be a power-aware middleware suitable for power-constrained devices.

Solar has similar objectives as ACAI. It defines a pervasive computing infrastructure for context-aware applications. Solar is built on abstracting context using direct acyclic graphs for collecting, aggregating, and disseminating context information. The abstract concept models context as events that flow through a directed acyclic graph of operators for processing and delivering context to subscribing events.

In comparison to ACAI, as the author mentions in [91], solar is not designed for ad hoc interactions and hence spontaneous applications. This is mainly because all services that are embedded in the Solar infrastructure like naming and discovery process are proprietary in nature. In addition, Solar supports scripting-type adaptation and no reasoning about context.

Theo suggested the mobile active space for personal mobile communication. He proposed the ACM and the MSK for modeling and inference about context and the XSP protocol for information exchange. The goals of his research were to some extend similar to what we propose, except that Theo did not use ontology concept for modeling context. Instead,

he used N3, an RDF based language, which only supports basic knowledge representation and lacks the richness of ontologies in modeling information and precisely those related to context. The inference method supported in his architecture is in the form of *is-a* relation, which does not capture the contextual information hidden in captured context, and cannot provide a good approximate reasoning like our proposed inference system, which is based on ontology and fuzzy inference methods. A final major difference is in the process of adaptation. Theo's XSP protocol uses explicit negotiation mechanism to exchange information among agents, while ACAI uses a multi-attribute negotiation mechanism to set the contextual information parameters expected by users and then automatically adapt itself according to these negotiated parameters and levels of context. An overall comparison between ACAI and other related research is shown in Table 2-1.

RESEARCH PROJECT	CONTEXT REPRESENTATION	INFERENCE SUPPORT	COMMUNICATION SUPPORT	ADAPTABILITY
EASY LIVING	XML	X	HTTP	X
COOL-TOWN	ATTRIBUTE-VALUE	X	HTTP	X
CONTEXT TOOLKIT	XML	X	HTTP	WIDGET RE-PROGRAMMING
GAIA	FOL	FOL	CORBA	USING SCRIPTS AND POLICIES
RCSM	CA-IDL	PROCEDURE	CORBA	CA-IDL
SOLAR	JAVA BASED NAMING	DAG	INS	FILTERING AND AGGREGATION
APMC	N3	RELATIONAL GRAPH	XSP	SERVICE NEGOTIATION
CONTEXT FABRIC	XML	X	HTTP	X
ACAI	ONTOLOGY BASED	FUZZY LOGIC AND DECISION TREES	HTTP	CONTEXT NEGOTIATION

Table 2-1 Comparison between ACAI and other projects

X means the functionality is not supported in the original architecture

2.5 The Context-Aware Conference Application

The background and related work explained in the previous sections give insight vision of the importance of context to applications in pervasive computing. This section clarifies this vision with a typical application that emphasizes the different aspects of context and our research perspective

2.5.1 Contexts in the Context-Aware Conference Application

One of the interesting things of being a researcher is attending conferences and symposia. Conference environments are loaded with services, activities, events, and human-computer interaction in its typical way.

We choose the conferencing scenario as the application our work will target because it holds within its nature a diverse amount of contexts that will highlight the importance of context in general and our contributions in specific.

Let us imagine that the next IEEE pervasive computing conference, Percom, is going to be the first context-aware conference. The organizers of the conference listed the different aspects that are required to provide context awareness and the different services and activities that will benefit from knowing context. After intensive discussions and meetings, they agreed that five contexts are required to be captured, interpreted, and manipulated for providing context-based conferencing services and activities. These contexts are ***Identity, Location, Time, Preferences, and Interests.*** We will call these contexts the group of five (Go5)

- **Identity**

The first and most important context is "**who**". Who is attending the conference? who is in the meeting room? who is presenting? who is next? who is the chairman? who is talking to whom? and so on. Thus, the next Percom will embed sensors in its venues that provide identity information. RF-Tags are the sensors used for providing this type of context. During the conference registration, which is nearly two month prior to conference starting time, users will provide their mailing address and Percom will mail

each registered user an RF-Tag to be used as his identification tag in the conference venue and any other conference-related places such as hotels and restaurants. Of course, attendees will return these RF-tags after the conference.

- **Location**

Knowing the identity is the first step towards context-awareness. However, knowing **"where"** is as important as **"who"**. Percom needs to know where attendee *x* is, where service *y* is, where device *z* is, where presentations will take place, where to forward notifications and announcements. Percom organizers decided to implement an indoor location system and chose infrared technology, Bluetooth and 802.11, in addition to the RF-Tag location sensors to provide this type of context.

- **Time**

The third context type addresses the situations that depend on time, **"when"**. When to start the presentation, when to send notifications and announcements, when to deliver a service, and so on. Percom implements a scheduling service where attractions, temporal events and time-dependent information are registered and stored.

- **Preferences**

Percom needs to know about the, **"how"**, preferences of the different services and devices expected to be in the conference venue. This information, unlike the other three described contexts, is hard to acquire using physical sensors. Thus, Percom had two solutions that are based on its semantic capability. The first is that registered users send the type and vendor name of the devices they have. Accordingly, Percom connects to the vendor's customer support site and maps the device capabilities to its ontology profiles. The second method is used to acquire this information by sending user-friendly ontology-based form to attendees to fill about their associated devices.

- **Interests**

The last context deals with the interests of the attendees, **"what"**. What food they like, what type of entertainments they favour, what are their research interests, and so on. Similar to preferences, interests are hard to acquire using physical sensors. During

conference registration, this information is acquired from forms sent to users to fill in and submit to the conference system.

We envision that the success of this Percom conference will motivate software developers to implement a system that stores this kind of information where it can be retrieved later for other conferences to use, hence reducing the burden of resubmitting the same information every time a user attends a conference. The next section addresses what Percom will do with these types of contexts in terms of the services and activities it is providing.

2.5.2 Services in the Context-Aware Conference Application

This section presents the services and applications that are enhanced with context in our imaginary Percom conference. We divided services into two categories. These are *"services in advance"* such as hotel booking, scheduling, and entertainment, and *"services on the fly"* such as instant messaging service, the conference physical resources and the event-notification service. The former category takes its input context from the attendees' profiles provided during conference registration while the latter takes its input context from the attendees' physical and logical context. Following is examples of these services.

- **Instant Messaging Service**

Percom offers its registered attendees an instant messaging service delivered to their handheld devices. This service provides features such as exchanging messages between attendees, projecting attendees' presence, delivering announcements issued by the conference organizers and more. The instant messaging also includes panel for browsing and requesting services in the Percom such as printers, projectors as well as methods for editing and modifying attendees' personal ontology profiles

Problem statement: attendees would like to know who is in the proximity. They do not want their presence projected to all users in the conference. They would like special type of announcement to be delivered to their devices. They would like some priority feature that can intelligently decide what to deliver in the instant messaging.

Proposal statement: Percom is aware about the context of the user. It knows the Go5 of each user, negotiates contextual information in order to adapt successfully to the expected Go5 of each user, reasons about these contexts, preserves user's privacy, and intelligently delivers announcements and news to users.

- **Service provisioning**

Attendees in the conference venue are nearly unaware of the available services. These services could be physical services or logical services. Some of the physical services are printers, projectors, wall screens, and email service, while some of the logical services are event notification, entertainments information, and attraction announcement services. Percom must support the ability to project these services to its registered users in such a manner that assures its compliance with users' requirements.

Problem statement: attendees are expected to invoke and interact with services in the conference venue. However, not all services are of interest to every user in the conference nor can the conference system scale gracefully if it projects all its services to all users.

Proposal statement: besides knowing the Go5 of the users, Percom is aware about the context of the available services. It can project services to users based on their location, activities, and services' capabilities.

2.5.3 Research Perspective

The context-aware conference scenario assisted us in shaping the perspective we took for developing an infrastructure for context-aware applications. This perspective was to explore the issues of formulating, reasoning, and adapting to contexts as our focus in the developing process of the infrastructure. These issues are summarized as follows

2.5.3.1 The Formularization Perspective

It is clear from the conferencing application that methods used to represent context are as important as methods of using context. Currently context is represented in an ad hoc, simple, and proprietary way. This method of representation lacks the concept of "*separating context formulization from its utilization*" and consequently, it hinders the

wide applicability of such systems. One reason why context is difficult to model comes from the diversity and complexity of this type of information. Context is collected from a wide range of heterogeneous sources such as sensors (hardware and software) and profiles while other contexts are derived from the aggregation of multiple sources and inference algorithms.

Ontologies are providing an interesting solution to interoperability, reusability, and domain sharing of concepts and relations. Accordingly, ontologies are adequate method for formularizing context and they facilitated other contributions in this thesis, such as reasoning about context and negotiating context.

Ontologies provide the extensibility and reusability concepts that are required in the conferencing scenario. For example, Percom models the Go5 of the printing service in the same manner as it models the Go5 of hotel service. Both use the same location ontology, the same user ontology, and the same service ontology but have different values for their classes, proprieties, and restrictions.

This thesis does not have the target to model semantically all types of context, as it is unrealistic, instead it models context in the form of *levels of expressiveness*. This method provides the sufficient knowledge in terms of classes and properties to model any common contexts found in the pervasive computing domain and at the same time provides the extensibility feature to specialize these contexts according to their applications' domain of discourse.

2.5.3.2 The Adaptation Perspective

Formalizing context is the first step towards a context-aware system. However, it does not make the system self-aware or adaptable. Adaptability requires the ability to reason about modeled context and to extract implicit contexts from already known ones.

The unavailability of adequate inference machineries to reason about context cripple current infrastructures and applications from evolving and reacting to hidden contextual situations that are more important than the captured contexts themselves [12]. For example, capturing the identity of a person through RF-tags, as in the imaginary Percom conference, without knowing the contextual relations this person has with other entities,

or privileges granted to that person will not provide the expected awareness required in pervasive environments. Thus, to have an adaptive infrastructure that supports context-aware applications, we need to develop adequate inference algorithms that use the power of semantically modeling context and are robust to the uncertainty and vagueness in captured context.

ACAI has the ability to reason about context using its inductive fuzzy inference component. This component applies induction and fuzzy reasoning to deduce actions, trigger events, and reason about context. In ACAI, context derives the induction process while applications and users provide the required conditions for controlling the inference process. These conditions are formulated as a negotiation protocol that runs between the infrastructure and the applications to decide the expected context types and their ranges that are needed by the applications and provided by the context providers in ACAI.

2.6 Summary

In this chapter, three interrelated fields of research have been discussed, namely context-awareness, pervasive computing and knowledge representation. Each of these fields was covered theoretically followed by a comprehensive survey of relevant research projects that are highly related to my work as well as to those fields. I presented a thorough comparison between the work proposed in this dissertation and the discussed projects. The comparison showed the advantages and drawbacks of each project and points that distinguished our work from these projects.

The research surveyed lead to the following observations:

- All related research incorporated context information in one way or another to achieve their intended purpose and to realize their goals.
- Related research suggested toolkits, middlewares, and infrastructures to ease the use of context when developing applications. However as there is no standard way of representing context most approaches excluded the use of the implicit semantics in captured contexts.
- In the literature so far, reasoning about context is based on logical reasoning. While this is acceptable in deterministic situations, there are situations where context is uncertain and this will introduce consequences that may not be as anticipated during system design
- Adaptation was taken into account in most of the projects and we enhanced this approach by negotiating context to allow for automated adaptation

Chapter 3

Agent-based Context-Aware Infrastructure

In Chapter 2, we discussed the different research perspectives that the thesis targets and the approaches employed to solve these research issues. In this chapter, we present the architecture of the proposed infrastructure, ACAI, which is developed to support the semantic modeling, the approximate reasoning, and the negotiation of contextual information in pervasive computing environments.

We start by stating the rationality behind using semantics and agents in developing ACAI, and the requirements that should be satisfied in this infrastructure. We then present the functional architecture that describes the functionalities provided by each component in ACAI. This is followed by the implementation architecture that describes the agents developed to realize the functional architecture. Lastly, we discuss the context negotiation mechanism that allows system adaptation based on the context requirements of entities in the environment.

3.1 Semantics and Agents: a Rational Debate

Interoperability and separation of concern are the major objectives of any computing infrastructure [116]. The application layer needs not to worry about interacting with the network layer and vice versa [28],[29],[30]. Application developers are relieved from managing, naming, controlling, and registering services and components with the infrastructure. In addition, the components of the infrastructure provide value-added services to the environment, such as event-notification, event-monitoring, service

discovery and more. However, the era of pervasive computing demolished the success of to date computing infrastructures. This is due to the heterogeneous resources and service personalization that pervasive computing is characterized with [4],[28].

To regain trust in delegating the management of entities in pervasive computing environments to the infrastructure, we need to investigate how the infrastructure can be more aware of the information it represents and more reliable in the services it provides. We also found that the interaction mechanism should be inspected and modified to cope with the new proposed context model. We used ontology engineering for modeling context and entities' preferences based on their contextual information [26],[22].

Ontologies define the semantics and common vocabulary of the domain used by entities that need to share information about that domain.[31],[32],[33],[34],[35]. They include machine-interpretable definitions of the basic concepts in the domain and the relations among them. Moreover, ontologies provide the ability to reason and deduce implicit information from the constructs defined in the ontology, reuse of domain knowledge, make domain assumptions explicit and separate domain knowledge from operational knowledge. Ontologies facilitated the following objectives in our infrastructure:

- Define the Go5 contexts explained in Chapter 2.
- Define the characteristics of services and negotiation mechanism.
- Describe hardware and software characteristics of resources.
- Represent supported network types and their preferences.
- Describe users' profiles such as contact information, scheduling, and activities.
- Define the common vocabularies used in fuzzy logic and inductive reasoning.

We used agents [21],[36],[37],[38] to realize the components of ACAI. This is because agents support autonomy, decision-making, behavioral structure, mobility and more. Agents are chosen over object-oriented programming because they have control over their behavior, i.e., they do not invoke methods upon inheriting one another but rather as requesting actions to be performed. This powerful concept is what our infrastructure needed for negotiating, inferring, and deciding action based on context. In addition,

agents are well suited for dynamic distributed environments, which is the default case in context-aware environments. Context could be geographically distributed, as in location-based information delivery; logically distributed, as in temporal events; or even both, as in spatial-temporal activities. Finally, agents have the ability to communicate and exchange information according to some shared and known ontology, which is exactly what we needed for exchanging context information.

3.2 ACAI Requirements and Goals

There are several requirements for the design of a context-aware agent-based infrastructure for pervasive environments. Some of the requirements mentioned here are general to any infrastructure development, while others are more specific to pervasive computing environments.

- **Transparency:** actions, inter-agent communications, and task accomplishments must be based on context captured from the environment. The infrastructure should at least adapt to the enduring activities, the capabilities of services, and the current situation of the users with minimal users intervention.
- **Expressiveness:** The infrastructure must support semantic and syntactic representation of context, actions, and rules that govern the infrastructure behavior.
- **Efficiency:** Inference and computing algorithms must be as efficient as possible.
- **Scalability:** The infrastructure must support many users and services simultaneously and without deep degradation in its behavior as result of overloading.
- **Robustness:** The infrastructure must be robust and overcome failures that may occur due to environment dynamism.
- **Seamlessness:** The infrastructure should provide users and devices with convenient access to a large set of services based on their requirements and context.
- **Reconfigurability:** The burden of setting up, invoking and discovering services, or giving instructions of operation to agents should be kept simple and to the minimum.

We believe that those requirements fall within the three objectives comprising our contributions. The first objective is to model semantically not only context but also rules and negotiation mechanism for setting contexts required by entities in the environment.

The second objective is persistency and inference. ACAI's core services must offer persistency through registration, lookup, and approximate reasoning mechanisms. The third objective, driven from the decision to use agent technology as ACAI's building technology, is negotiating context [112] information between the client side (applications and users) and the infrastructure side for setting clients' required quality of context and for the adaptation of ACAI's behavior. Finally, ACAI APIs must provide mechanisms for manipulating contextual information and managing applications running in the pervasive environment.

These objectives are addressed throughout the thesis. We will start by presenting the architecture of the overall system followed by every objective in details.

3.3 ACAI Functional Architecture

ACAI functional architecture represents the logical components needed for collecting, processing, inferring and disseminating context information in context-aware applications [15]. These components have the goal of facilitating the seamless interaction with the surroundings while hiding the complexity inherent in managing heterogeneous sources that provide context information.

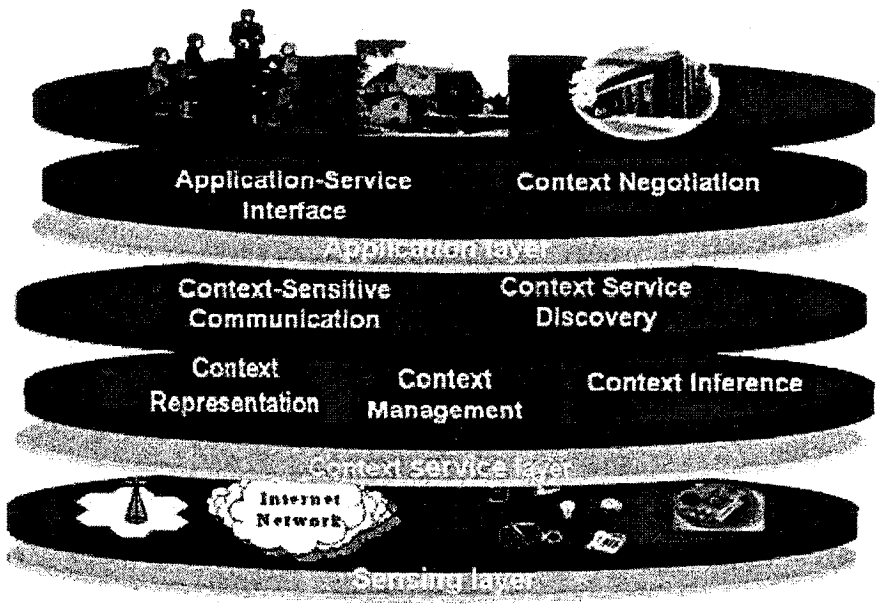


Figure 3-1 The functional architecture of ACAI

ACAI's functional architecture achieves the concept of "separation of concern" via its layered architecture shown in Figure 3-1. In the first layer, context is sensed and captured by diverse types of sensors and context sources embedded in the environment. In the second layer, the captured context is interpreted and structured by the context representation module. The context inference module uses this processed context for deducing other contextual information that cannot be explicitly sensed by the first layer. With these two modules, expressiveness and efficiency is achieved. However, applications and users using ACAI need methods to invoke and use this context. ACAI provides functionalities such as context-based service discovery that provides awareness about services available in the environment, such as a printing service, according to location and services' preferences. It provides a context management service for storing,

querying, and accessing context of users and resources with limited capabilities, and finally a context-sensitive communication protocol for event notification, service and information delivery, and presence projection. These functionalities realize robustness, seamlessness, and scalability requirements. The third layer is primarily an interface to the underneath layer for negotiating context information between applications and context providers that are using ACAI as their infrastructure. With this layer, the remaining requirements stated in section 3.2 are satisfied in the ACAI infrastructure.

ACAI layers can be summarized as follows:

3.3.1 The Sensing Layer

The sensing layer is responsible for capturing and acquiring context information about the artifacts found in the environment. It consists of different sensors, software agents, and network probes that cooperate to monitor different context information. Captured context information is then passed to the context service layer. Functionalities provided by this layer are one of the thesis assumptions stated in Chapter 1.

3.3.2 The Context Service Layer

The context service layer has two goals. First, representing captured context in a well-structured, unified, composable, and extensible model. Second, providing service provisioning based on context.

The first goal is achieved using the ontology module for representing context and the inference module for reasoning about context. The inference module reason about captured context in order to deduce and induce new context information that is undetectable by sensors, such as contextual relations and dependencies.

The second goal is achieved through three functional modules: the context management module, the context-sensitive communication module, and the context-based service discovery module. The context-based service discovery module provides the capability of discovering services and resources to users and applications based on their context. It is a peer-to-peer discovery process where each peer node holds information about services, users, and even context of interest from peer's point of view. To provide seamless

information access, context-aware applications need to interact with existing devices, services, and users. The context-sensitive communication module allows entities to communicate using unique identifications via application-level addressing. The context-sensitive communication module provides presence, notification, context-based network connectivity within the same domain and across different domains.

The context management module is addressed in this thesis and will be discussed in subsequent chapters. However, the other two modules are considered to exist in the infrastructure and provide the functionalities required from them as we stated in the assumption section in Chapter 2.

3.3.3 The Application Layer

This layer provides the interface between users and applications on one side and ACAI's core services on the other. It also includes the negotiation mechanism that allows ACAI to adapt to applications' requirements of context information [22],[25]. This layer addresses the following tasks:

- A declarative method with which users state their context requirements and context providers offer dynamic customized context-based service information
- Protection against unexpected performance degradation because context providers know users' needs and can thus better manage their resources
- Seamless context-based services across domains in a large environment
- An easy mechanism with which context providers can intelligently supply and monitor context resources.

In the next section, I will present the agent architecture that realizes the proposed components of the functional architecture. I will discuss each agent in terms of its role, its interaction with other agents and the supported functionalities that can be invoked using this agent. The agent architecture will give more clarification to the reason of realizing ACAI as a multiagent model.

3.4 ACAI's Agent Architecture

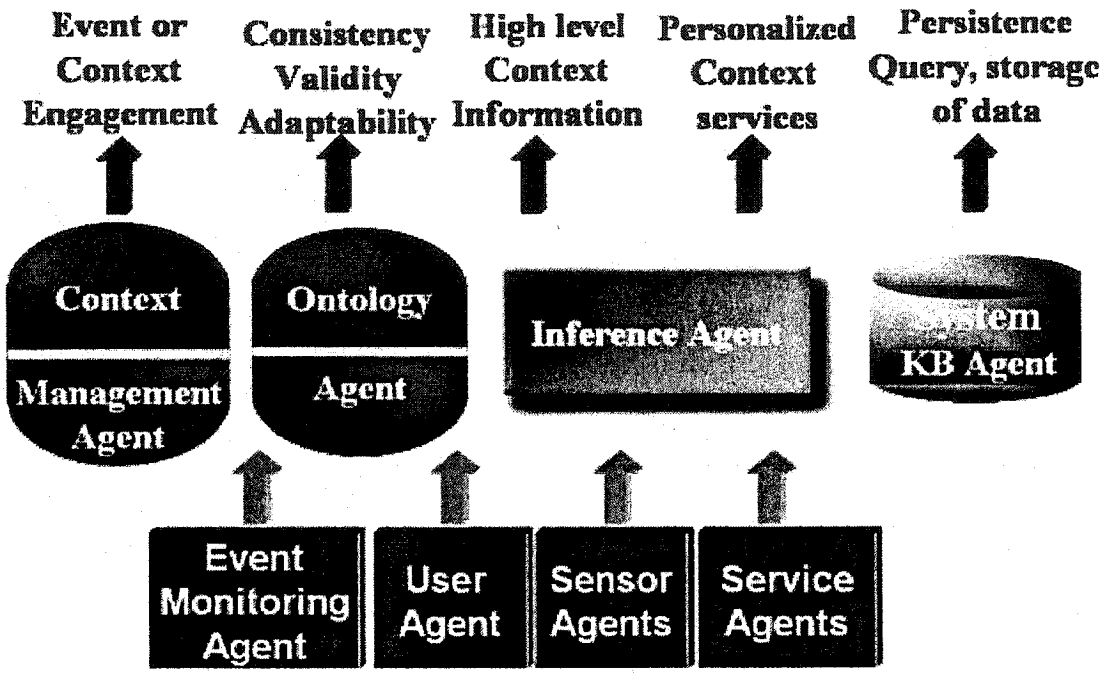


Figure 3-2 ACAI agent architecture.

ACAI's agent architecture, shown in Figure 3-2, represents the concrete implementation of the functional architecture we discussed in section 3.3. The Context Management Agent (CMA), the Ontology Agent (OA), the Inference Agent (IA) and the System Knowledgebase Agent (SKBA) are considered the heart of ACAI's implementation.

CMA is responsible for managing the contextual information of entities in the environment, for interacting with the SKBA in order to assign, load, modify, and delete information stored in the system repository and for enforcing rules that govern ACAI's behavior. CMA is also responsible for managing ACAI's ongoing sessions and used resources in the environment in order to achieve fairness in resource allocations and to guarantee the required contexts that were negotiated for during session initialization. CMA also monitors changes in context to capture any violations in agreed on contexts and to resolve any conflicts that may occur.

SKBA wraps the repository system of ACAI. It provides methods for storing ACAI's rules, user/service profiles, service execution conditions, agreed-on contextual information during session setup and methods to semantically match and retrieve stored

information and ontologies. The ontology agent (OA) provides the semantic capability of the Infrastructure. It models both context and meta-context information using ontologies implemented in OWL. The Inference Agent (IA) performs the inductive fuzzy reasoning to deduce implicit contextual information and to insure that fired actions are consistent with the negotiated contexts. The coming subsections discuss each agent in more details.

3.4.1 Context Management Agent (CMA)

The context management agent is the administrator of the system. It is responsible for monitoring ongoing sessions and managing the environment resources. CMA has the authority to cancel, modify, and renegotiate context specifications according to changes that might occur in the environment, or if agents violate the tasks assigned to them during negotiation.

Agents register with the CMA to project their presence to other agents in the environment and to obtain the authorization needed to negotiate and use context information. The CMA stores relevant information in the knowledgebase repository for inference, consistency maintenance, and knowledge sharing. This repository, which is controlled by the system knowledgebase agent (SKBA), is used for retrieving information of entities in the environment and for the temporal storage of session information. CMA accepts subscription requests from users, services, and context provider agents to register them in the agent directory facility. This subscription phase gives CMA the ability to search and select the best available context providers that offer context information required by applications and users in the pervasive environment. In addition, CMA binds CPAs to applications according to their context agreements and stores this binding information in the system knowledgebase for the dynamic reconfiguration of resources as context changes.

Figure 3-3 shows an excerpt of the internal structure and methods supported by the developed CMA. The *action()* method is the actual behavior of the CMA; it defines the way CMA processes messages received from and sent to other agents in the environment as well as the conditions for invoking other methods in the CMA. The *calculatexx()* method is used for monitoring the time taken in the negotiation process and for deciding

other time-related actions. The *SendtoAgent()* is the method used for sending context information and any other required information from CMA to other agents in the system. The *Subscribexx()* is used to register agents with CMA for discovery and availability issues. The *Bindxx()* is used to logically bind agents that have common features or actions together. For example, it is used to bind CPAs and UAs after negotiation. The *Propose(), Reachxx() and Request()* are mainly used during the negotiation phase for setting contexts of interest to entities in the environment under the control of CMA.

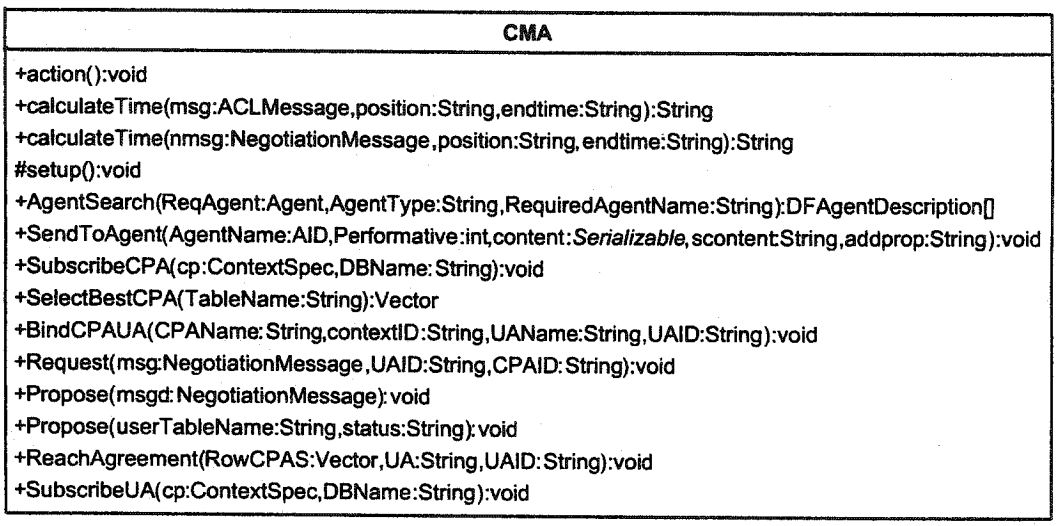


Figure 3-3 Excerpt of the Context Management Agent internal structure

3.4.2 Ontology Agent (OA)

The ontology agent provides the semantic capability of ACAI. It models context and system information using ontologies that are implemented using OWL. At negotiation time, it formulates context specifications from agents' requests and profiles based on the designed ontology constructs, axioms, and rules of composition that we will discuss in the coming chapter. This information is then provided to the CMA for further processing and to the SKBA for storage and manipulation. The OA also provides search and browsing interfaces so that registered agents can lookup and use available ontologies. In addition, the OA is responsible for the validation, consistency checking, and mapping between ontologies used in ACAI.

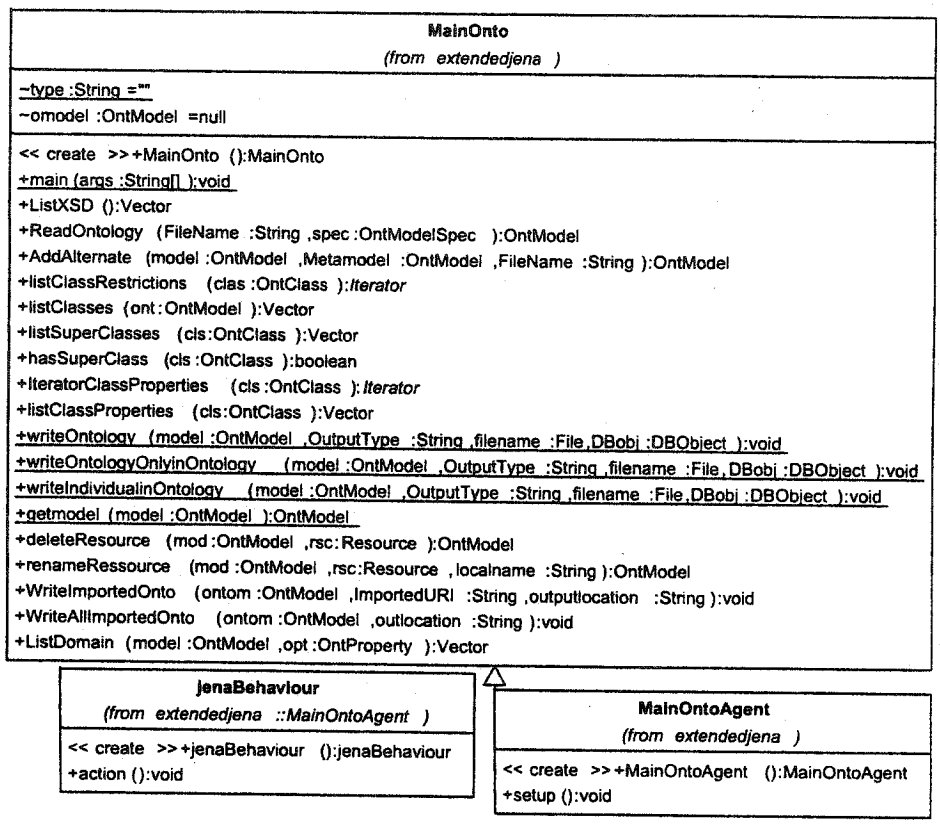


Figure 3-4 Excerpt of the OA internal structure

Figure 3-4 illustrates the different methods that are supported by the OA. The OA supports all the required methods to manipulate ontologies starting from reading ontologies, passing by the incremental modification of the triples in the ontology graph and ending with writing ontologies to the repository or to some output stream.

One of the contributions in the OA implementation is its ability to read ontologies and instances from different media and merging them for manipulation, then writing each one individually to different output streams if needed. This feature is not available in current open sources for ontology manipulation such as protégé or Jena. Also, as we will discuss in the context modeling chapter, the OA supports new meta-information concepts like the *Similarity* and *Alternatives* for matching and merging ontology constructs [26].

3.4.3 The Inference Agent (IA)

The IA supports an inference mechanism that integrates logical reasoning, fuzzy reasoning, and semantic rule representation in one system. Contexts captured from

sensors and from users and services' profiles are treated as facts in the context inference process. The inference agent uses these facts, which are stored in the system knowledgebase repository, to deduce new context information and trigger actions. The IA also uses logic-reasoning mechanisms to insure that captured context instances are consistent both with each other and with arguments defined in the context ontology.

The contribution of using fuzzy inference to reason about context is supported in the IA. This contribution is needed to cope with the vagueness in context and the uncertainty in actions to be taken. The IA uses fuzzy approximate reasoning in general and the inductive fuzzy reasoning in specific, which is discussed in details in Chapter 5.

Figure 3-5 shows an excerpt of the methods and internal classes built in the inference agent. The *FetchAgent* is a behavior class that is executed during negotiation and service discovery situations to match available context provider agents (CPAs) and service agents (SAs) to applications and users requests. The *BuildTree* class has the required methods to implement the inductive fuzzy reasoning engine and to infer about context and context-related information at run-time.

The IA is also responsible for resolving conflicts arising from new facts and rules, for runtime validation, and consistency checking of context information. It is also responsible for ensuring that changes made to an entity's information, as result of reasoning, would not result in a violation or inconsistency to other entities in the environment.

In the implantation chapter, I will demonstrate some applications, such as the instance messaging application, the service provisioning application, and the event-notification application, where the inference mechanism was used to deduce actions and higher-level context information.

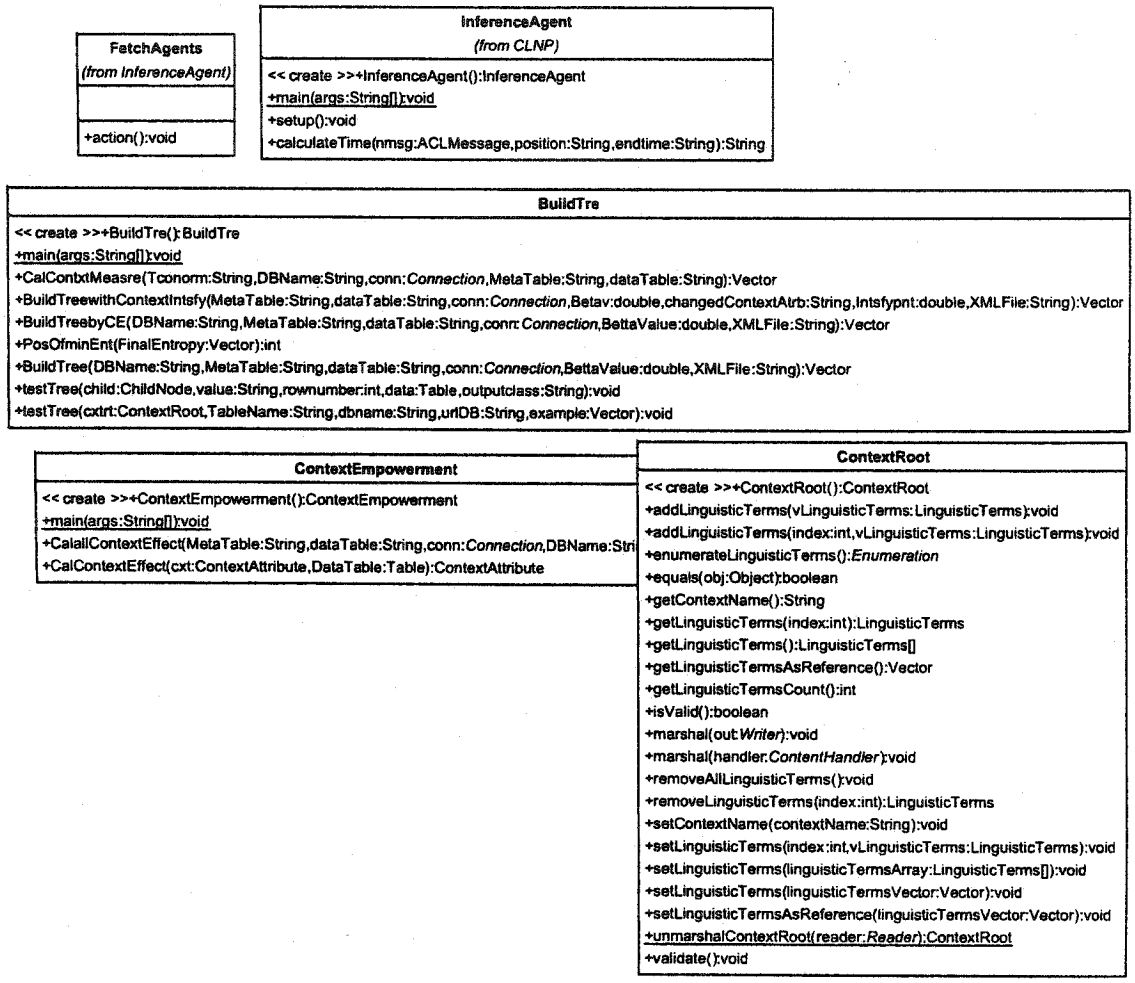


Figure 3-5 Excerpt of the inference agent internal structure.

3.4.4 System Knowledgebase Agent (SKBA)

Using ontologies for representing context information require a storing facility and a manager for this facility. The SKBA is responsible for managing this repository space. More precisely, it acts as the interface to the repository where queries, in the form of RDQL, about stored information are sent and appropriate responses retrieved from the repository are replied back.

The repository also includes tables for storing the binding information between context providers and entities requesting context information, tables that record response time,

number of messages exchanged in ongoing sessions and tables of registered agents and their roles in the environment. All these tables are managed and manipulated through the SKBA.

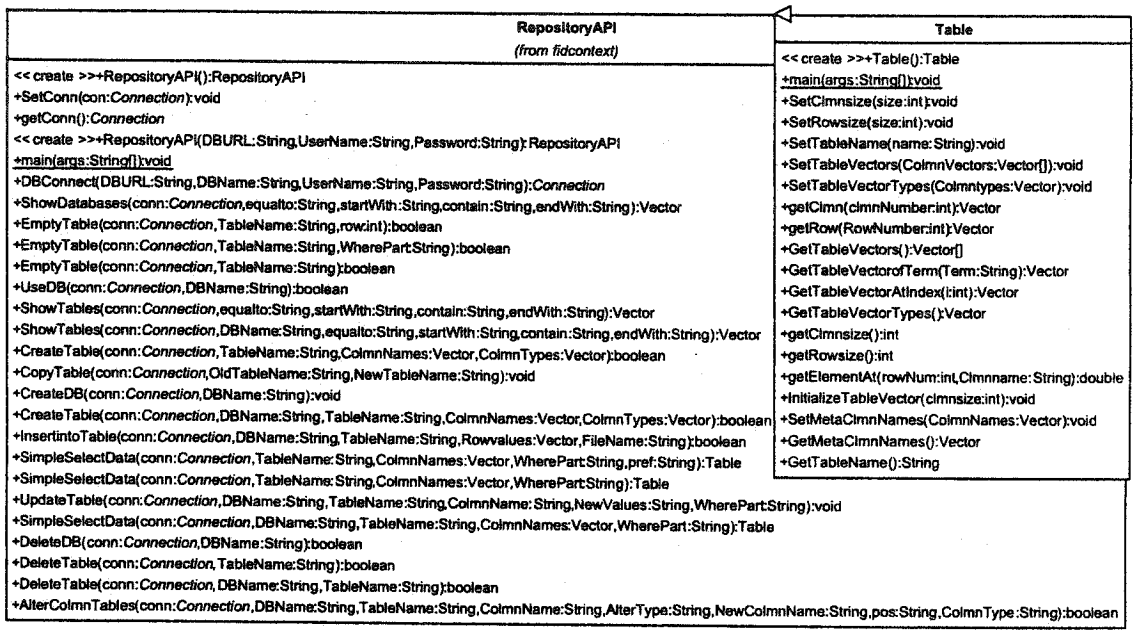


Figure 3-6 Excerpt of the SKBA internal structure

Figure 3-6 shows an excerpt of the methods found in the developed SKBA. Two behaviors are implemented in SKBA to support its functionalities. The first behaviour is the *RepositoryAPI*, which is dedicated to handle the process of creating, querying, and deleting context information from the system repository. The second behaviour is the *TableAPI*, which is concerned with the process of controlling and manipulating meta-information about the repository space. It provides methods for annotating meta-information about the repository space such as table names, columns types, ontology namespaces, and more as depicted in Figure 3-6.

3.4.5 The Context provider agent (CPA)

Each context source in the environment is wrapped with a context provider agent. This agent is responsible for capturing raw data from the source and mapping it to its ontological representation. In case the CPA is unable to perform the mapping, it communicates with the ontology agent to do the mapping of captured context information

to its semantic representation. The mapped context is stored in the system knowledgebase so that other agents can engage using the lookup service that the ontology agent provides. Furthermore, the CPA is responsible for negotiating context specifications acquired by the context source under its control and for monitoring context sessions that they are involved in as illustrated in Figure 3-7.

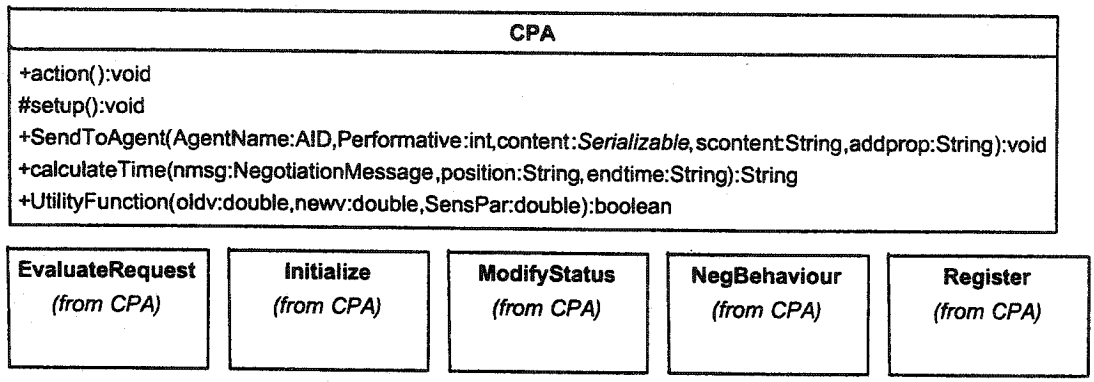


Figure 3-7 Excerpt of the CPA agent internal structure

3.5 Enabling Adaptation through Context Negotiation

In the previous sections, we described the functional and agent architecture of ACAI to support context-aware applications in pervasive environments. In this section, we describe the interaction mechanism among these agents and show how we modelled it in the form of a negotiation process. This negotiation allows ACAI to adapt to the changes in the evolving situations and to the requirements of entities that developers cannot anticipate at design time.

In pervasive computing, adaptation is an essential feature to cope with the many forms of spontaneity and uncertainty in context information. Therefore, the infrastructure should provide the ability to inspect and control context provisioning according to applications requirements at runtime. As a result, we developed a negotiation mechanism to allow entities to specify, and agree on, the types and values of contexts that they expect to have from the context provider agents registered with ACAI.

Examples of context negotiations are customizing services to users, methods of representing context, and methods of event notification. This distinguishing feature of negotiating context gives ACAI the ability to adapt its behaviour by reacting to the changing environment conditions and taking into account user factors such as location, service requests, service description, and more. This has the advantage of overcoming the static nature of context provisioning in current research projects, discussed in Chapter 2, which may not necessary be the most appropriate method to deal with the changes in the environment situations. However, the thesis is not concerned with theoretical formulation and analysis of the strategies in negotiation protocols. Rather, it is concerned with the process with which agents evaluate and generate values for context information when involved in certain activities.

In general, negotiation is a process by which a joint decision is reached between two or more agents, each trying to maximize an individual goal or objective. The agents first communicate their positions, which might conflict, and then try to move towards agreement by making concessions or searching alternatives [115],[117].

Major elements of any negotiation process are

1. The language used by the participating agents.
2. The protocol followed by agents as they negotiate.
3. The decision process used to determine agents' positions, concessions, and criteria of agreement.

I will present how our proposed negotiation process defines each of those elements along with an example for clarification.

3.5.1 A Negotiation Ontology as a Common Language

As I will present in Chapter 4, the designed ontologies include a category that defines meta-information, and one of these meta-information ontologies is the negotiation ontology. This ontology is the semantic language to be used among agents in ACAI, in order to have a common understanding about the performatives, the utility functions, the negotiation rules, and the contextual information used in the negotiation.

The negotiation ontology imports the fuzzy ontology that we will discuss in Chapter 5 to define the rules that govern the negotiation and imports all developed context-related ontologies in order to define the context information that the agents will negotiate. The ontology is reasonably generic and does not impose any restrictions on the performatives or the utility functions used. However, we created instances of those classes with our specific performatives and utility function that ACAI will use as a default. This provides ACAI with the extensibility required to cope with the diverse situations that it can be used in and imposes no restriction on the methods of interactions among agents. Figure 3-8 shows an excerpt of the classes and properties defined in the negotiation ontology.

- CLNP (hasContextSpec*, hasManager*, hasProcess*, hasProvider*, hasRequester*, usesUtilityFn*)
- ContextParameter (hasType*, hasUtilityValue*, usedInUtilityFn*)
- ContextSpecification (hasContextElement*,hasContextParameter*)
- NegotiationProcess (TimeToLive*, hasRule*)
    - NegotiationMessage (inResponseTo*, messageType*)
        - instance CLS_Proposal
    - NegotiationRules (hasOperand*)
- UtilityFunction (minRange*, maxRange*,hasSensitivityparameter*)
    - LogFn ()
- SoftwareAgent ()
    - CLNP_Actors (involvedInCLNP*)
        - CMA ()
        - CPA ()
        - Consumer ()
            - SA (hasService*)
            - UA (hasUser*)
        - IA
- Condition ()
    - RuleOperand (hasAntecedent*, hasConsequent*, operandName*)
        - instance and
- Antecedent ()
- Consequent ()

Figure 3-8 Excerpt of the negotiation ontology

CLNP is the main class of the negotiation ontology[1]. It is used as the root element of any negotiation instance to define the different contextual parameters required in the negotiation process. The property *hasContextSpec* has the class *ContextSpecification* as its range. This class links the negotiation ontology to any other context-related ontology.

[1] http://www.schemaweb.info/schema/SchemaDetails.aspx?id=181

The ContextParameter class defines the ranges of the contextual information under negotiation. Its effect in the negotiation process is defined using the *hasUtilityValue* property. The property *hasProcess* is used to express the interaction of the negotiation in terms of the negotiation messages used, and the negotiation rules that govern this process.

```
<CLNP:CLNP rdf:ID="CLNP_khedr">  <!-- start of the negotiation instance -->

  <CLNP:hasRequester>
   <CLNP:UA rdf:ID="khedrAgent"/>   <!-- The agent who requested the negotiation -->
  </CLNP:hasRequester>

  <CLNP:hasContextSpec>   <!-- the context specification that is used in the negotiation -->
   <CLNP:ContextSpecification rdf:ID="Location"
     CLNP:hasContextElement="&Loc;PlaceInstance"> <!-- reference to location ontology-->
    <CLNP:hasContextParameter rdf:resource="#DeliveryDelay"/>
   </CLNP:ContextSpecification>
  </CLNP:hasContextSpec>

  <CLNP:hasProcess>     <!--the negotiation process has negotiation message of type
                              CLS_Proposal and also has Rule1 that governs the process-->
   <CLNP:NegotiationMessage rdf:ID="CLS_Proposal"
     CLNP:messageType="CLS_Proposal"
     CLNP:TimeToLive="20ms">
    <CLNP:hasRule>
     <CLNP:NegotiationRules rdf:ID="Rule1">
      <CLNP:hasOperand rdf:resource="#and"/>
     </CLNP:NegotiationRules>
    </CLNP:hasRule>
   </CLNP:NegotiationMessage>
  </CLNP:hasProcess>

</CLNP:CLNP>

<CLNP:RuleOperand rdf:ID="and"
   CLNP:operandName="And">
                          <!-- the rule that is used in the negotiation with two antecedents
                               and one consequent -->
  <CLNP:hasAntecedent>
   <CLNP:Antecedent rdf:ID="bigger_than"
     CLNP:hasValue="5ms">
    <CLNP:hasContextParameter>
     <CLNP:ContextParameter rdf:ID="DeliveryDelay">
      <CLNP:usedByYtilityFn>
       <CLNP:UtilityFunction rdf:ID="LogUF"
         CLNP:hasSenstivityparameter="5">
        <CLNP:hasContextParameter rdf:resource="#DeliveryDelay"/>
       </CLNP:UtilityFunction>
      </CLNP:usedByYtilityFn>
     </CLNP:ContextParameter>
    </CLNP:hasContextParameter>
   </CLNP:Antecedent>
  </CLNP:hasAntecedent>

  <CLNP:hasAntecedent>
   <CLNP:Antecedent rdf:ID="less_than"
     CLNP:hasValue="10ms">
    <CLNP:hasContextParameter rdf:resource="#DeliveryDelay"/>
   </CLNP:Antecedent>
  </CLNP:hasAntecedent>

  <CLNP:hasConsequent>
```

Figure 3-9 Excerpt of a negotiation instance

For example, in the instance messaging application presented in Chapter 6, a user may negotiate with the CMA about the delay that may be encountered before changes in location is projected to the user. This negotiation instance is depicted in Figure 3-9 that shows the initiator of the negotiation (*KhedrAgent),* the type of message used to exchange this negotiation (*CLS_Propose*), and the rule that governs the negotiation process (*Rule 1*). In the shown example, the rule states that if the delay is between 5 and 10 ms then the user is not accepting the offer and will renegotiate by sending a counter proposal with new values.

3.5.2 The Negotiation Formalization

In this section, we describe the negotiation process in terms of the types of messages exchanged between ACAI agents and the content of these messages. This is followed by the utility function calculation that is used in evaluating agents' concessions and decision-making.

The developed negotiation process is more like an interaction mechanism between the agents to define ranges, domains and types of context required by entities on whose behalf they work and comply with the constructs defined in the context ontologies that we will present in Chapter 4. Thus, agents do not compete to win the negotiation process but they cooperate to maximize their understanding of context.

3.5.2.1 Types of Messages Exchanged

Six types of messages can be exchanged between ACAI's agents. Each serves a specific purpose in the negotiation process and each has its required conditions that need to be satisfied in order to send it from one agent to another. The types of messages are:

- **CLS_Negotiation**

When an agent wants to share its understanding of context or to request a service from ACAI according to some context specifications, the first message to send is the CLS_Negotiation. This message has user/service agents as the senders and CMA as the receiver. It has the following formal format

CLS_Negotiation:: { Sender, Receiver, Content, Properties}

Sender:: {UA ∨ SA}

Receiver:: {CMA}

Content:: {CLNP_instance}

CLNP_instance:: {context_specifications, utility_values, negotiation_conditions}

Properties:: {deadline_time, message_sequence_Id}

- **CLS_Proposal**

The CMA to reply to a CLS_Negotiation or a CLS_Offer message uses this type of message. The CMA evaluates the context specifications of the CLNP instance of the received CLS_Negotiation and matches between these specifications and those that are advertised by CPAs registered with the CMA. The outcome of the evaluation is the content of a CLS_Proposal. The CLS_Proposal has the following formal format

CLS_Proposal:: { Sender, Receiver, Content, Properties}

Sender:: {CMA}

Receiver:: { UA ∨ SA}

Content:: {CLNP_instance}

CLNP_instance:: {context_specifications, negotiation_conditions}

Properties:: {deadline_time, message_sequence_Id}

- **CLS_RePropose**

The initiator of the negotiation might not be satisfied by the returned results, which are sent from the CMA in a CLS_Proposal message. His satisfaction is based on utility function calculation that I will discuss in the coming section. If this is the case, the initiator will send a CLS_RePropose to the CMA with the modified context specifications and their new utility function values. CLS_RePropose has the following formal format

CLS_RePropose:: { Sender, Receiver, Content, Properties}

Sender:: {UA ∨ SA}

Receiver:: {CMA}

Content:: {CLNP_instance}

CLNP_instance:: {context_specifications, utility_values, negotiation_conditions}

Properties:: {deadline_time, message_sequence_Id}

- **CLS_Request**

If the CMA receives a CLS_RePropose, it will send to the available CPAs a CLS_Request message where it requests if any of them can provide the new proposed values. The CLS_Request will have the modified context specifications received from the initiator but not their new utility function values to preserve entities privacy. CLS_Request has the following formal format

CLS_Request:: { Sender, Receiver, Content, Properties}
Sender:: {CMA}
Receiver:: {CPAs}
Content:: {CLNP_instance}
CLNP_instance:: {context_specifications, negotiation_conditions}
Properties:: {deadline_time, message_sequence_Id}

- **CLS_Offer**

Once they receive a CLS_Request, CPAs will evaluate the context specifications found in its content and decide, based on the utility function calculation, if they can provide these specifications and the amount of concessions that they can sacrifice. The result is wrapped in the content of a CLS_Offer and sent back to the CMA. The CLS_Offer will have the modified context specifications and their new utility function values. CLS_Offer has the following formal format

CLS_Offer:: { Sender, Receiver, Content, Properties}
Sender:: {CPAs}
Receiver:: {CMA}
Content:: {CLNP_instance}
CLNP_instance:: {context_specifications, utility_values, negotiation_conditions}
Properties:: {deadline_time, message_sequence_Id}

- **CLS_Agree**

The last type of messages is the CLS_Agree that is used to end and close the negotiation process. As we mentioned, the negotiation process is more like exploring the context specification supported by ACAI CPAs and not a competence method, the negotiation always ends with a CLS_Agree. The CLS_Agree will have the final agreed on context specifications. CLS_Agree has the following formal format

CLS_Agree:: { Sender, Receiver, Content, Properties}

Sender:: {CMA, UA, SA, CPAs}

Receiver:: {CMA, UA, SA, CPAs}

Content:: {CLNP_instance}

CLNP_instance:: {context_specifications}

Properties:: {message_sequence_Id}

3.5.2.2 The Message Flow of the Negotiation Process

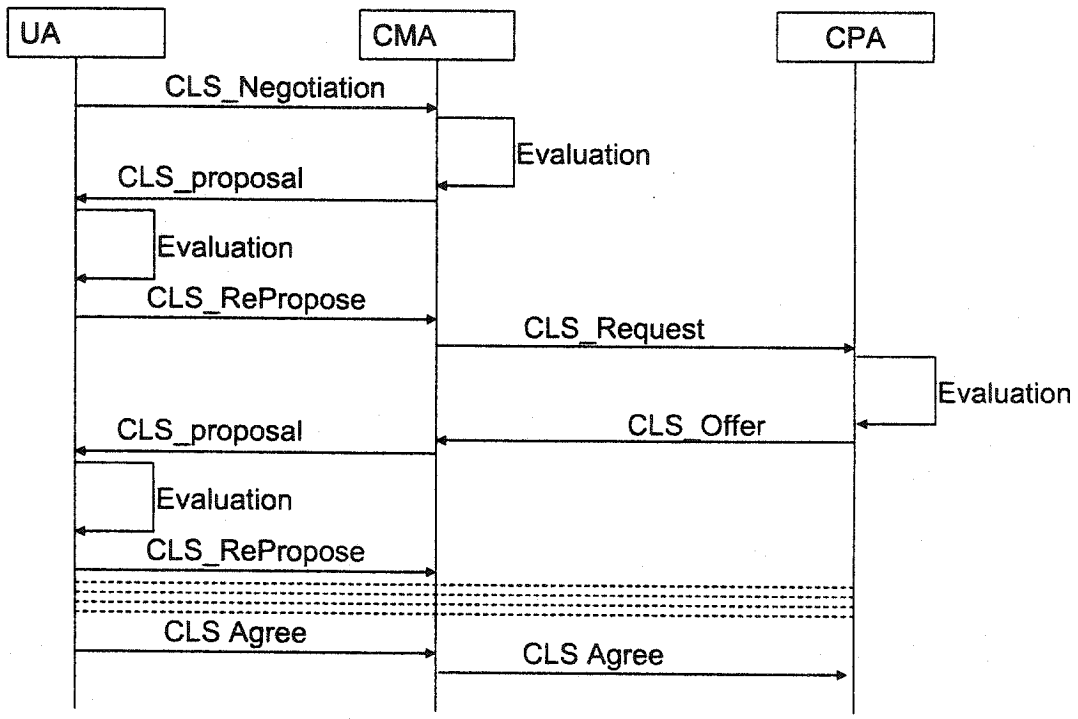


Figure 3-10 Message sequence diagram of the negotiation protocol

In general, protocols are used to coordinate the activities of a group of agents as they try to cooperate in performing certain tasks. In particular, this section specifies the protocol between ACAI's agents looking for an agreement about context specifications. Figure 3-10 gives the sequence diagram of the messages exchanged in the protocol.

Negotiation always starts from the applications' side, i.e. the user or service agents, with the CLS_Negotiation message. The CMA is the intermediary agent in the negotiation protocol and is responsible for monitoring and controlling the negotiation interaction mechanism. Upon receiving a CLS_Negotiation message, it evaluates the required context specifications and replies with the new values to the initiator in a CLS_Proposal message. If these values are acceptable by the initiator, the negotiation terminates,

otherwise the initiator sends a CLS_Repropose with the modified specifications to the CMA again. In this case, the CMA invokes the CPAs registered with it to decide if they can make concessions on their offered specification values.

Based on the contents of the received CLS_Request message, CPAs evaluate their acceptable concessions and inform the CMA with these values and their corresponding utility function values. The negotiation process is iterated after that until either an agreement is reached or the negotiation dead line time expires. In the latter case, the negotiation terminates with the implicit agreement of using the last values of the context specifications being negotiated and exchanged between the agents.

We intentionally designed the negotiation process to be as simple as possible since it occurs in a dynamic environment that is highly limited in resource capabilities. The type of the utility functions used in the negotiation process also contributes to this simplicity. In the next section, I will present the logarithm utility function that ACAI's agents use in the negotiation process.

3.5.2.3 Utility Function

Entity's contexts, such as presence information in an instant messaging application or availability information for accessing services in a conferencing application, are the types of specification that are negotiated for. In other words, users should be allowed to project their required contexts to the infrastructure, as they are the only ones who know what their goals are at any moment and how different contexts are valued.

Utility functions serve this purpose. A utility function U_i: $C \rightarrow \Re^+$ translates entity i's required contexts, $\{1..j\}$, into $u_i(C_j)=u_{i,j}$ that represent how much that entity values these contexts within their negotiated range.

The required utility functions that are used in context negotiation must satisfy the following characteristics:

- A lower bound on x (context parameter). Below this value, user satisfaction is negligible and is approximated to zero.
- An upper bound on x. Above this value, any gain in user satisfaction is negligible and is approximated to one.

- Between these two bounds, satisfaction factor is a monotonically increasing function of x.
- A sensitivity parameter "p" for projecting user satisfaction.

We decided to choose the logarithmic function as it matches these requirements. The utility function will have the following form:

$$U(x) = 1 - a.\ln(b.x + c) \quad \ldots\ldots 1$$

$$a = \frac{1}{p - Ao} \quad \ldots\ldots \quad \ldots\ldots \quad \ldots\ldots \quad .2$$

$$b = \frac{e^{1/a} - 1}{R - A} \quad \ldots\ldots \quad \ldots\ldots \quad \ldots\ldots \quad 3$$

$$c = \frac{R - Ae^{1/a}}{R - A} \quad \ldots\ldots \quad \ldots\ldots \quad \ldots\ldots 4$$

Where U(x) = utility function.

x = context parameter values in negotiation.

p= sensitivity parameter.

A_o= expected context sensitivity.

A= minimum context parameter value.

R= maximum context parameter value.

The utility function shape depends on the sensitivity parameter "*p*". For every value of "*p*", entities project their interest in the context specification according to the utility function shape as shown in Figure 3-11. For example for "*p*" values smaller than A_o, entities are more sensitive to larger values of the context parameter than to smaller values, such as location information in outdoor applications. For "*p*" values larger than A_o entities are more sensitive to smaller context values than to higher values, such as delay in emergency notification time.

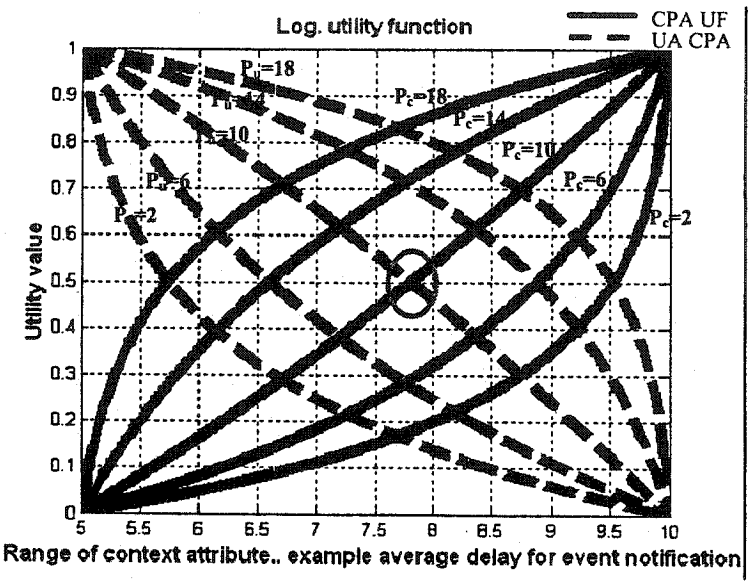


Figure 3-11 Utility functions that are used in the negotiation process

The user/service agent decides the working utility curve based on the current negotiation process and types of context used in the negotiation. The utility curve is decided by applying the following decision equations on the initial context parameter value A_L requested by the user/service agent.

$$U(A_L)_{x=A_L} = a.\ln(b.A_L + c)$$

$$If \quad U(A_L) << U(R) \text{ for } \quad A_L < R, \ \& \quad A_L > \left[\frac{A+R}{2}\right] \quad ....5$$

then $P < A_o$, agent is more sensitive to large context values and "p" is chosen from equation 2 to satisfy equation 5

$$If \quad U(A_L) >> U(A) \quad for \quad A_L > A, \& \quad A_L < \left[\frac{A+R}{2}\right] \quad .....6$$

then $P > A_O$, agent is more sensitive to small context values and "p" is chosen from equation 2 to satisfy equation 6

The same process applies to the context providers involved in the negotiation. The only difference is in the shape of the utility function as illustrated in Figure 3-11. CPAs have the inverse equation of the utility functions. This is because CPAs tend to preserve their resources. An agreement is expected when both curves intersect as illustrated in Figure 3-11 with the green circle area.

3.6 Example

In this section, we illustrate how the proposed negotiation mechanism is used to exchange the ranges of contexts that are relevant to the applications and should be known to ACAI agents. In particular, we refer to our conferencing application and present an example of how context negotiation can be successfully applied to explore, share and set contexts between ACAI agents.

A service that our conference application can customize is the event notifications delivered to entities (users and services) based on their location, scheduling, and information availability contexts. Events could be about physical entities such as the existence of printers, presence of users or logical such as the starting/ending time of a meeting, changes in devices preferences and network capabilities.

In particular, application developers at design-time model these contexts using the developed context ontologies. Then at runtime, user/service agents negotiate with the CMA and CPAs the expected ranges of these contexts that they value the most.

Figure 3-12 shows an excerpt of a negotiation instance example. In this example, agents negotiate the range values of the location and delay-time contexts used to project the availability of services and are modeled using the ontologies discussed in Chapter 4. The example shows two place instances, (B502, A620), which are used in the notification service, the data type which is supported at those two locations, (Cartesian type), and the levels of context that the agents can negotiate (same level, domain level and best effort). Each of these levels has different context ranges. When users subscribe to the notification service, they can negotiate with CMA the expected ranges of these two contexts that suit them and their situations in the environment. The default values for these contexts are applied to users if they do not wish to negotiate.

```
<rdf:RDF>
   <!--    THE LOCATION DATA USED IN THE EVENT-NOTIFICATION SERVICE    -->
   <Loc:LocationData rdf:ID="LocCartesian">
      <Loc:hasUnit>meter</Loc:hasUnit>
      <Loc:hasRadius>10</Loc:hasRadius>
      <Loc:min>0</Loc:min>    <!--  THE MINIMUM RANGE TO NEGOTIATE   -->
      <Loc:max>10</Loc:max>  <!--  THE MAXIMUM RANGE TO NEGOTIATE   -->
   </Loc:LocationData>
   <Service rdf:ID="LocationService"> <!--    START OF THE SERVICE DESCRIPTION    -->
      <presents>
         <ConProf:Profile rdf:ID="LocationServiceProfile">
            <ConProf:textDescription>
            provide event notification based on location
            </ConProf:textDescription>
            <ConProf:serviceName>LocationService</ConProf:serviceName>
            <serviceParameter>
   <!--THE SAME LEVEL NOTIFICATION.... OTHER TYPES ARE DOMAIN AND BEST EFFORT -->
               <ConProf:ServiceParameter rdf:ID="samelevel">
                  <ConProf:serviceParameterName>
                  SameLeve
                  l</ConProf:serviceParameterName>
               </ConProf:ServiceParameter>
            </serviceParameter>
         </ConProf:Profile>
      </presents>
      <Loc:hasPlaceInstance rdf:resource="&Loc;B502"/>
      <Loc:hasPlaceInstance>
         <Loc:Room rdf:ID="A620">
            <Loc:hasLocationDataType>
               <Loc:CartesianType>
                  <Loc:hasLocData rdf:resource="#LocCartesian"/>
               </Loc:CartesianType>
            </Loc:hasLocationDataType>
         </Loc:Room>
      </Loc:hasPlaceInstance>
   <!--    REST OF THE LOCATION SERVICE    -->
   </Service>
   <!--    THE AVAILABILITY SERVICE    -->
   <Service rdf:ID="AvailabilityService"/>
   <!--    REST OF THE AVAILABILITY SERVICE    -->
</rdf:RDF>
```

Figure 3-12 Excerpt of the contextual information used in the negotiation

Now suppose that a user initiates the negotiation process to set the ranges of the location that is of interest to him. These ranges and other information are shown in Table 3-1. The table shows that the user, *UA0,* negotiates for location ranges (between 5 and 8 meters) that can be used to project his presence to other agents in the surroundings.

ContextSpec	UAName	UA_ID	CParName	Cpar_min	Cpar_max
Location	UA0	UA0-6204520	ProjectionRange	5	8

Table 3-1.The parameters used in the negotiation

When the CMA responsible for managing these two locations receives the CLS_Negotiation, it will query the repository to fetch CPAs that provides location information with ranges matching user's context specifications in the CLS_Negotiation message. The output of the query is shown in Table 3-2.

ContextSpec	CPAName	CParName	COS	Capacity	Cpar_min	Cpar_max	unit
Location	CPA1	ProjectionRange	DomainLevel	15	6	10	m
Availability	CPA11	Av.Delay	MediumDelay	8	5	12	s

Table 3-2 Returned results for two agents negotiating contexts with the CMA

As shown in Table 3-2, *CPA1* matches to some extent (inexact) what *UA0* specified in his negotiation request. Thus, the negotiation process continues to set a mid-ground on the values of contexts that satisfy the user's requirements and can be provided by the CPA. The whole negotiation process is shown in Table 3-3.

UAID	AgentInAction	TypeOfMessage	StartTime	EndTime
UA0-6204520	UA0	CLS_Negotiate	1087604661050	1087604661060
UA0-6204520	CMA	ACL_Query	1.0876E+12	1.0876E+12
UA0-6204520	Infa	ACL_Inform	1087604661230	1087604661230
UA0-6204520	CMA	CLS_Proposal	1.0876E+12	1.0876E+12
UA0-6204520	UA0	CLS_RePropose	1087604664405	1087604664405
UA0-6204520	CMA	CLS_Request	1.0876E+12	1.0876E+12
UA0-6204520	CPA1	CLS_Offer	1087604670864	1087604670864
UA0-6204520	CMA	CLS_Proposal	1.0876E+12	1.0876E+12
UA0-6204520	UA0	CLS_Agree	1087604685225	1087604685225
UA0-6204520	CMA	CLS_Agree	1.0876E+12	1.0876E+12
UA0-6204520	CPA1	CLS_Agree	1087604695489	1087604695489

Table 3-3 The complete negotiation message flow

Accordingly, after reaching an agreement, the CMA binds *UA0* to *CPA1* and stores this information in the knowledgebase repository for monitoring, evaluating, and adapting the notification service according to these context values.

3.7 Summary

In this chapter, we presented the ACAI functional architecture and its concrete implementation using a multi-agent system.

As explained in this chapter, ACAI relies on the sensing layer to deliver its basic functionalities, in particular, context-based services. On top of this layer is the context service layer, which consists of a set of components that implements the ontology model, the context management model, and the inference mechanism.

We described the developed multi-agent system, the roles supported by every agent in the system and briefly discussed methods, and behaviours implemented in these agents.

ACAI is tailored to applications needs by adapting its behaviour to their required contexts. These contexts are monitored by ACAI based on the negotiation occurred between the applications and ACAI.

Finally, we discussed the context negotiation process that allows ACAI to adapt its services and provide customized context information based on users' requirements and expectations.

The demonstrated example showed the importance of modeling context and meta-context information ontologically as it facilitated the process of negotiation since agents will share a common understanding about contexts and only tune these contexts to their own requirements

In the subsequent chapters, we discuss in depth the core components of ACAI. In particular, we discuss the context ontology module and the management and inference module, followed by the prototype implementation and performance evaluation.

Chapter 4

Semantic Representation of Context

Context is any information about the surrounding environment, physical as well as logical. This information when captured creates the implicit interaction between humans and computers [27].

Context has even a greater impact when we advance into non-traditional, off-the-desktop computing environments. Thanks to mobile and pervasive computing, users expect to be able to access whatever information and services they want, any time, and everywhere [6],[39]. With computers used in such wide variety of situations, interesting new challenges rise. This lead to the invention of infrastructures and middlewares to facilitate the use of context and to enhance the development of context aware applications [40],[41],[42],[43].

In general, the focus was to consider predefined situations and ad hoc representations of context to address the process of developing context-aware applications and delivering services in pervasive environments. However, this ad hoc approach makes it hard to build new applications or even improve existing ones. This is because the ad hoc representation of context limits context reusability across applications, and impels developers to redesign common functionalities found in context-aware applications. Moreover, appropriate logic for using and reasoning about context has to be worked out. This causes new context-aware infrastructures to be redeveloped from the scratch.

To address these issues, I will present in this chapter the types of context considered in this thesis, the semantic modelling of context using ontologies, and the management of these ontologies.

4.1 Ontology-based Context Information

Ontologies represent a new paradigm for modeling and representing contexts that are needed in pervasive computing [44],[45],[46]. They have the feature of representing the semantics in the "what, where and when" information necessary for the development and management of context-aware applications.

One concern that we faced during ontology design was what contextual information to model. This was largely due to the overwhelmingly diverse types of context found in pervasive computing environments. We reached a conclusion that completely formalizing all context information is unrealistic. Instead, we found that it is more appropriate to model context in the form of ***Levels of Expressiveness***, shown in Figure 4-1, where the highest level is an abstraction of all concepts of context and as we go down the levels, context is expressed in more details and refined to a representation that is more concrete. [15]

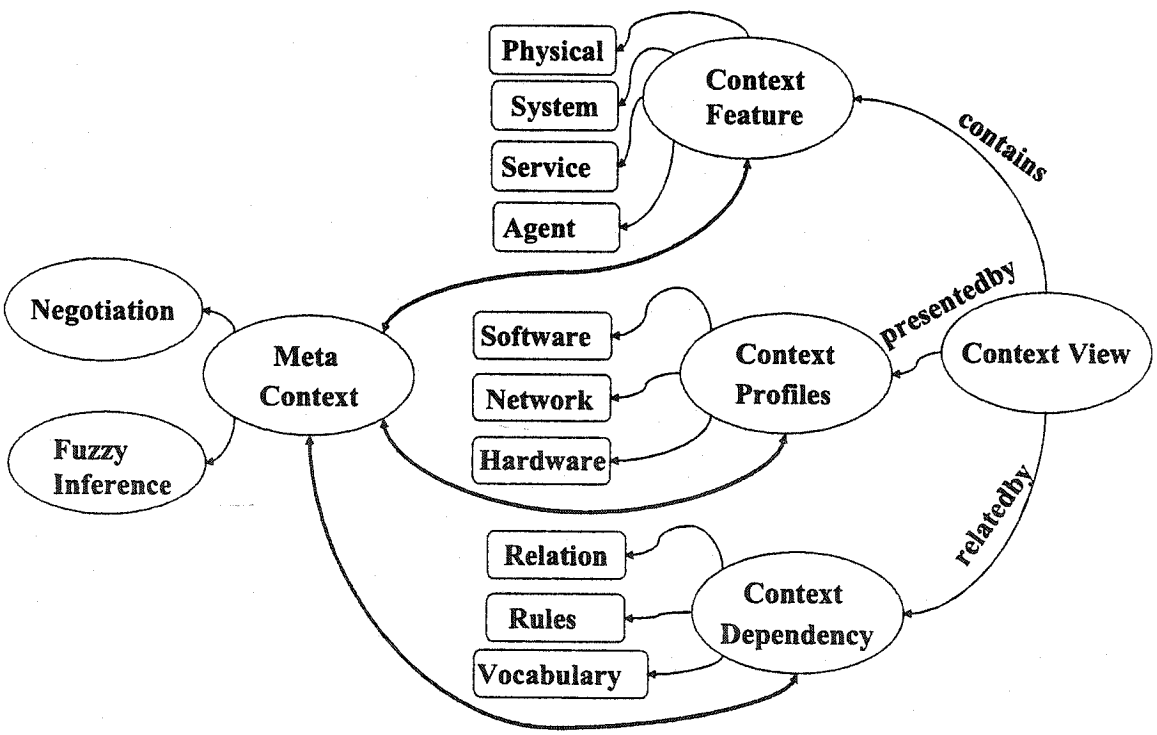


Figure 4-1 The developed conceptual ontology for context-aware environments

The highest level of abstraction in the ontology structure is the *ContextView* that provides an organizational point of reference for declaring context information. *ContextView*

represents the different types of contexts that belong to a certain entity. An entity could be a user, a service, or even the environment itself. At least one instance of the *ContextView* will exist for each distinct entity in the environment.

The properties *contains*, *presentedby* and *relatedby* are properties of the *ContextView*. The classes *ContextFeature*, *ContextProfile* and *ContextDependency* are the respective ranges of those properties. These classes are considered the second level of expressiveness in the ontology. Each instance of the *ContextView contains* one or more *ContextFeature* classes that are *relatedby* one or more *ContextDependency* classes and *presentedBy ContextProfile* classes.

The *ContextFeature* class models the Go5 context information we discussed in Chapter 2. It consists of sub-classes that are related to the physical environment (such as location), classes that are related to the system (such as services), classes that are related to persons (such as actors), and classes that are related to the underlying network (such as network characteristics).

Profiling is the way of representing the extent to which something exhibits various characteristics. Profiling will represent the characteristics of devices and services in the environments. The *ContextProfile* will represent the services and users preferences and ACAI's networking characteristics. Profiling has the "separation of concern" feature, i.e. what ACAI, as an infrastructure, requires as data input is separated from how it uses this data.

The *ContextDependency* class relates the other two classes together, i.e. the *ContextFeature* class and the *ContextProfile* class. It defines rules of composition, rules of engagement, and relations that must hold between composed context information. Rules are defined either in the form of logic rules or fuzzy rules to cope with the ambiguity and vagueness in context information.

As Figure 4-1 shows, we also modeled the meta-context information of the infrastructure in the form of two ontologies, the fuzzy ontology, and the negotiation ontology. This is a novel concept in context modeling, as current research tends only to model the data of the system in the form of ontologies but not the meta-data that glue these ontologies together or define the behavior of the system.

In the coming section, I will discuss the ontology development life cycle. After that, I will present the different ontologies that we designed to model context and the three perspectives that we took in the design. These perspectives are why we need this ontology, what context is modeled in it, and how it is used in a context-aware application.

I presented the negotiation ontology in the previous chapter. I will defer the discussion of the fuzzy ontology to the next chapter since it is more related to the concepts presented in that chapter.

4.2 Ontology Development Life Cycle

It is well-known that there is no distinct methodology for developing ontologies [47] and that several versions of the ontology can, to some extend, model the same domain correctly. This is because the ontology development life cycle is an iterative process as shown in Figure 4-2. Ontology developers start with a rough first-pass at the ontology concepts and terminologies, then revise and refine the evolving ontology, and finally fill in the details.

Ontology experts have agreed on fundamental rules that any developer must be aware of and follow in his ontology design. These rules are as follows.

- Concepts in the ontology should be close to objects (physical or logical) and their relationships in the domain of interest. These are most likely to be nouns (objects) and verbs (relationships) in sentences that describe the domain.
- There is no single way to model a domain correctly using ontologies. There are always viable alternatives. The best solution usually depends on the application that the developer has in mind and the extensions that he anticipates.
- Ontology development is essentially an iterative process.

We thus summarize the steps required in the ontology development life cycle as follows:

a) Decide the domain that the ontology is addressing and its levels of detail. This will guide many of the modeling decisions down the road.

b) Among several viable alternatives, determine which one would work better for the required domain and be more intuitive, more extensible, and more maintainable.

c) Ontology is a model of reality of the world and the concepts in the ontology must reflect this reality.

d) Evaluate and debug the ontology by using it in simple applications.

e) Revise the ontology.

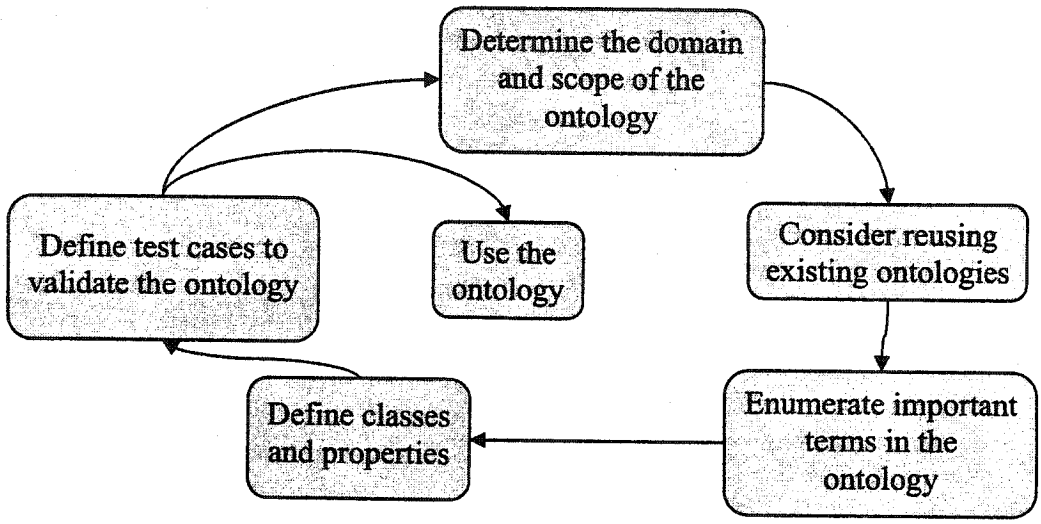


Figure 4-2 Ontology development life cycle

This process of iterative design will likely continue throughout the entire life cycle of the ontology and even after the ontology usage.

The first step in the life cycle is elaborated by our concept, ***Levels of Expressiveness***, which is used to model the granularity of context information. We start by defining roughly the general contexts used in the domain of pervasive environments and as we go down the road, we express context in more details and specialize the ontology to particular situations.

The other steps are elaborated separately in the coming sections that describe every type of ontology we designed.

4.3 The Location Ontology

Location is considered the most important information in any context-aware application. This is because any interaction with the users or the applications requires knowing their location whether physically or logically. In addition, any other context information, such as activities, actions and service provisioning, is highly dependent on location. The problem lays in the diverse nature of location information and the nonexistence of valuable ontologies to reuse as proposed by the second step of the ontology lifecycle. Thus, we modeled our own location ontology, excerpt shown in Figure 4-3, and took a global perspective in all our designed ontologies. That is not only to model the intended domains of the ontologies, but also to model their meta-information in order to facilitate extensibility and reusability requirements in the design of ontologies.

4.3.1 Context Information in the Location Ontology

All Ontologies in ACAI include constructs that represent the contexts they are modeling and others that address the extensibility and reusability requirements. This distinguishes us from any available ontology, if it actually exists, that might represent to some extend the same context information.

We solved the problem of diversity in the location information by separating location information into two types: *ContextualLocation* and *PlaceInstance*. The first type models the low-level location information generally acquired from location sensors. The second type abstracts this low-level information to logical location context. The reason behind using this approach is to maximize the generalization of ontology usage as new constructs can be added in any of the two types without affecting already defined constructs and without causing any major rework in the ontology classes and properties.

These two classes are related together with the properties: *hasRelationToPlace* and *hasLocationDataType* respectively. Both properties are inverse properties, i.e. instantiating one will automatically instantiate the other. We defined in the *ContextualLocation* class the most commonly used location data types. In particular, we defined GPS, Cylindrical, Cartesian and Address data types.

Under the *PlaceInstance* class, we categorized places into indoor or outdoor places. These categories have the property *hasPlaceDescription* that can compose one place instance from other places thus facilitating the process of describing complex places at run time. Another key class is the *SpatialEntity*, which is used as an anchor to bind location information to other contextual information such as people and services.

- Similar ()
  - LocationSimilar ()
- ContextualLocation ()
  - Address (hasCountry*, hasStreet*, hasStreetNumber*, hasZipCode*)
  - LocationData (hasAccuracy*, hasAddress*, hasAltitude*, hasDirection*, hasHeight*, hasHorizontal*, hasLatitude*, hasLongitude*, hasNetworkAddress* hasRadius*, hasRelativeLocationData*, hasSeta*, hasUnit*, hasVertical*)
  - LocationDataType (hasAlternativeTrans*, hasLocData*)
    - AddressType ()
    - CartesianType ()
    - CylindricalType ()
    - GPSType ()
    - LogicalType ()
      - NetworkType ()
    - VectorType ()
  - NetworkAddress (hasDomain*, hasIP*, hasSubnet*)
- LocationRelation (hasObject*, hasPredicate*, hasSubject*)
  - PhysicalRelation ()
- Place ()
  - IndoorPlace ()
  - OutdoorPlace ()
- PlaceInstance (hasLocationDataType*, hasLocationRelation*, hasPlaceDescription*, hasUniqueID*)
  - Building (hasFloor*)
  - Complex ()
  - Floor (belongstoBuilding*, hasRoom*)
  - GeoRegion ()
  - Park ()
    - CarPark (haslot*)
    - Recreational ()
  - Room (belongstoFloor*)
  - UserDefinedPlace ()
  - sightSeeing ()
- SpatialEntity ()
- TemporalEntity ()
  - LocationRole ()
    - ComplexRole (hasLocationRole[2+])
    - Educational ()
    - Entertainment ()
    - Personal ()
    - Social ()
    - UserDefinedRole ()

Figure 4-3 Excerpt of the location ontology.

The *LocationRole* class specifies location-based activities. The preferences of these activities are defined in the activity and activity-profile ontologies. The *LocationRole* is a subclass of the *TemporalEntity*, defined in the time ontology, since activities are time-dependent as well as location dependent. This will give the ability to associate temporal information to activities running in an ACAI pervasive environment.

4.3.2 Meta-Information in the Location Ontology

Meta-information is what distinguishes our location ontology from similar existing ones. By meta-information, we mean information that improves ontology reusability, enhances its mapping, and facilitates its extensibility.

To improve reusability, the location ontology has the *LocationSimilar* class, which is subclass of the *Similar* class in the fuzzy ontology that we describe in the next chapter. Each class in the location ontology has the property *hasSimilarity* and its range is set to the *LocationSimilar* class. This concept allows reusing the location information of one place as the location information of another place and specifies the degree of similarity between them in the form of a fuzzy membership function.

For extensibility support, we defined the *UserDefinedPlace*, which is an abstract class for developers to extend and define place instances that are not currently supported in the location ontology. This class is used along with the property *hasPlaceDescription* in order to describe any new place instance in terms of existing instances. Also, the *ComplexRole* class is used to represent user-defined roles as the conjunction of other location roles.

The last type of meta-information addresses the mapping requirement. This is realized using the property *hasAlternativeTrans* and the class *LocationRelation*. The *hasAlternativeTrans* property is used to define methods of transformation between different location data types. The *LocationRelation* class is used to describe how one place instance can be mapped to another using the general tuple <subject, predicate, object>.

4.3.3 Example of Location Ontology

Figure 4-4 illustrates an example of the location ontology that elaborates its key features. The figure shows two place instances: *MMARL* of type *UserDefinedPlace* and *B502* of type *Room*. Both instances represent the same place but they are defined by different developers. To state that they are similar, the *LocationSimilar* class is used. This example shows that *MMARL* and *B502* are similar with a discrete fuzzy membership function that has value equal to *true* and are similar in terms of *LocationInformation*, which is an instance of type *PhysicalRelation* class.

```
<GPSType rdf:ID="MMARLGPS">
    <!--this property is used to state the relation among locations-->
    <hasRelationToPlace rdf:resource="#MMARL"/>
    <!--this property is used to map between locations-->
    <hasAlternativeTrans rdf:resource="#B502Address"/>
    <hasLocData>
        <LocationData rdf:ID="GPSData">
            <hasLatitude>45.5</hasLatitude>
            <hasLongitude>47.8</hasLongitude>
        </LocationData>
    </hasLocData>
</GPSType>
<!-- A place instance that the developer defined as user-defined. Example of ontology extensibility-->
<UserDefinedPlace rdf:ID="MMARL">
    <hasSimilarity rdf:resource="#MMARL-B502"/>
    <hasLocationDataType rdf:resource="#MMARLGPS"/>
</UserDefinedPlace>
<!--a place instance of type room-->
<Room rdf:ID="B502">
    <hasLocationDataType rdf:resource="#B502Address"/>
</Room>
<!--example of ontology reusability-->
<LocationSimilar rdf:ID="MMARL-B502">
    <Fuzz:similarTo rdf:resource="#B502"/>
    <Fuzz:similarIn rdf:resource="#LocationInformation"/>
    <Fuzz:degreeOfCertain>
        <Fuzz:Discrete rdf:ID="SureSame">
            <Fuzz:hasInputValue>True</Fuzz:hasInputValue>
            <Fuzz:hasMemebeshipValue>1.0</Fuzz:hasMemebeshipValue>
        </Fuzz:Discrete>
    </Fuzz:degreeOfCertain>
</LocationSimilar>
<PhysicalRelation rdf:ID="LocationInformation"/>
<AddressType rdf:ID="B502Address">
    <hasLocData>
        <LocationData rdf:ID="AddressB502">
            <hasAddress>
                <Address>
                    <hasStreet>king Edward</hasStreet>
                    <hasStreetNumber>800</hasStreetNumber>
                    <hasZipCode>K1N6N5</hasZipCode>
                </Address>
            </hasAddress>
        </LocationData>
    </hasLocData>
</AddressType>
```

Figure 4-4 example of location ontology

The figure also shows that *B502* has a *ContextualLocation* instance of the *AddressType* class while *MMARL* has a *ContextualLocation* instance of the *GPSType* class. Since both of them represent the same place, it is expected that their location information is also similar. This is stated in the example using the *hasAlternativeTrans* property.

4.4 The Time Ontology

- TimeFormat ()
  - FormalType (hasTimevalue*)
    - Day ()
    - Hour ()
    - Minute ()
    - Month ()
    - Second ()
    - StandardTime ()
    - Year ()
  - InformalType (associateFormalFormat*)
    - DayRelated ()
      - Afternoon ()
      - Morning ()
      - Night ()
      - Noon ()
      - evening ()
    - YearRelated ()
- TemporalEntity (hasTemporalRelation*, firstEntityInRelation*, restEntityInRelation*, hasTimeParameter*)
- Continuous ()
  - TemporalDescription (hasUnit*, hasTemporalRelation*)
- Discrete ()
  - TemporalDescription (hasUnit*, hasTemporalRelation*)
- TemporalRelation ()
  - instance after
  - instance before
  - instance contains
  - instance during
  - instance equals
  - instance finishedBy
  - instance finishes
  - instance meets
  - instance metBy
  - instance overlappedBy
  - instance overlaps
  - instance startedBy
  - instance starts
  - instance within

Figure 4-5 Excerpt of the time ontology

Time and temporal relations play an important role in context-aware environments, as most actions, activities, and interactions are time-dependent. Research in [49] proposed a time ontology that defines time instances, time intervals, and relations among temporal events. It defines time as crisp values in the form of calendar time and clock time. This time representation is quite adequate for traditional desktop applications but we faced problems of vagueness and time-impreciseness when we tried to use it in pervasive environment applications. Sessions, activities, service-requests, event-notifications, and all other types of time-dependent applications are loosely coupled to crisp time information. We successfully managed to overcome this obstacle by importing our developed fuzzy ontology into the time ontology and use it to give us the ability of relaxing the crispness in time information.

4.4.1 Context Information in the Time Ontology

The time ontology, shown in Figure 4-5, consists of three main classes: *TimeFormat*, *TemporalRelation* and *TemporalDescription*. The *TimeFormat* is an abstract class that defines the different time formats supported in our ontology. It has as subclasses the *FormalType* class that describes time in the standard format YYYY:MM:DD-HH:MM:SS and the *InformalType* class that represents time similar to the way it is used in human conversation. The *TemporalDescription* class defines how this informal type of time is modeled as a fuzzy variable with a membership function describing the points of the interval associated with it. The *TemporalRelation* class relates time intervals and/or temporal entities together using one of the defined relations in the ontology. The defined relations are the standard Allen's temporal relation but extended with the fuzzy capability.

4.4.2 Example of the Time Ontology

Meetings are good example of temporal entities that have impreciseness in their time information. Meeting organizers are aware of the fact that no meeting conforms to its expected time slot and scheduling is the main annoyance in any venue that has multiple consecutive and parallel sessions. If one session exceeds its assigned time slot, puzzlement rises immediately on how to re-arrange the rest of the sessions and every

coming session contributes to the scheduling puzzle. However, our time ontology with its fuzzy capability can greatly reduce the burden of uncertainty in time intervals. Figure 4-6 illustrates this concept. The example shows a meeting instance that has the type of a *TemporalEntity* class and ID equals meetingA. This meeting coexists with another *TemporalEntity*, meetingB. This relation is modeled using the *TemporalRelation* "*during*". The example shows that meetingA takes place at 12:05 on January the 27th. This timing information is fuzzified with a Gaussian MF that has a mean equal to 12.05 and variance set to 5.0. This fuzzification means that the starting time of *meetinA* is now relaxed 5 minutes from 12:00 to 12:10 with each instance having a membership value taken from the Gaussian MF. Thus, the organizers and the inference agent in ACAI can take into consideration this relaxation in the timing information to set and schedule other meetings in the environment.

```
<rdf:RDF>
    <TemporalEntity rdf:ID="MeetingB"/>
    <TemporalEntity rdf:ID="meetingA">
        <firstEntityinRelation rdf:resource="#meetingA"/>
        <hasTemporalRelation rdf:resource="#during"/>
        <restEntityInRelation rdf:resource="#MeetingB"/>
        <hasTemporalDescription rdf:resource="#timeOfmeetingA"/>
    </TemporalEntity>
    <TemporalDescription rdf:ID="timeOfmeetingA">
        <hasTemporalRelation rdf:resource="#starts"/>
        <hasTimeParameter rdf:resource="#meetingtime"/>
        <Fuzz:hasAssociatedFuzzyTerm rdf:resource="#meetingAfuzzyterm"/>
    </TemporalDescription>
    <StandardTime rdf:ID="meetingtime">
        <hasTimevalue>2004-01-27T12:05:00   </hasTimevalue>
    </StandardTime>
    <Fuzz:FuzzyTerm rdf:ID="meetingAfuzzyterm">
        <Fuzz:hasMembershipFn>
            <Fuzz:Gaussian rdf:ID="GausmeetingA">
                <Fuzz:hasmean>12.05</Fuzz:hasmean>
                <Fuzz:hasVariance>5.0</Fuzz:hasVariance>
            </Fuzz:Gaussian>
        </Fuzz:hasMembershipFn>
    </Fuzz:FuzzyTerm>
            <TemporalRelation rdf:ID="during"/>
            <TemporalRelation rdf:ID="starts"/>
</rdf:RDF>
```

Figure 4-6 Example of a time ontology

4.5 The Actor Ontology

The main objective of pervasive computing is to manage appropriately the interactions between users and the environment. Thus, it is essential to model users' context and to model the way they interact with the environment. Modeling of users must follow the same perspective that ACAI tries to achieve, which is the semantic modeling of all entities in the environment in order to provide a shared common understanding of the structure of information among entities and to enable the reuse of domain knowledge.

4.5.1 Context Information in the Actor Ontology

Following the life cycle of ontology development and in particular the "consider reusing existing ontologies" step, we did not model user's context from scratch. Instead we used the "Friend Of A Friend" (FOAF) ontology [50] which is a collaborative effort amongst Semantic Web developers to semantically model user's information.

The FOAF project is based around the use of machine-readable web pages for people, groups, companies, and other kinds of a Thing. To achieve this, the "FOAF vocabulary" provides a collection of basic terms that can be used in these web pages. The initial focus of FOAF has been on the description of people, since people are the things that link together most of the other kinds of things described on the Web: they make documents, exchange mails, attend meetings, and are depicted in photos.

FOAF defines classes such as foaf:Person, foaf:Group, foaf:Image, and foaf:Document, and some useful properties of those classes, such as foaf:name, foaf:mbox (i.e. an internet mailbox), foaf:knows, and foaf:homepage. In addition, FOAF defines some useful relationships that hold between members of these classes. For example, the property foaf:depiction that relates a foaf:Person to a foaf:Image.

However, FOAF cannot be used "as is" in context-aware applications. This is because FOAF is designed for web-based personal information rather than pervasive-based personal information. FOAF does not have the notion of software agents that work on behalf of persons. Furthermore, it does not provide the ability to relate persons to other

entities in the environment such as devices, services and activities based on context especially location and time. Thus, we extended the FOAF ontology with extra classes and properties in order to make it suitable for context-aware applications.

- SpatialEntity ()
  - Agent (hasActivity*, hasContactInfo*, hasDevice*, hasSimilarity*, made*, performs*)
    - Group ()
    - Organization ()
    - Person (hasAgent*, family_name*, firstName*, hasProfile*, knows*, publications*, topic_interest*)
    - SoftwareAgent (preferences*)
- WebPresenceID (idValue*, nickname*, registeredTime*)
- AddressType ()
  - AgentContactInfo (DefaultHomepage*, PersonalHomepage*, RelevantHomepage*, mbox*, phone*, workplaceHomepage*)
- Similar ()
  - AgentSimilarity ()
  - Document (siteURL*, name*, topic*)
- Image (imageURI*)

Figure 4-7 The modified FOAF ontology

Figure 4-7 shows an excerpt of the modified FOAF ontology, Confoaf, in terms of its classes and properties. The first thing we made was to subclass the *SpatialEntity* in our location ontology with the *Agent* class from the FOAF ontology in order to associate agents to places. This provided the *Agent* class with the properties defined in the location ontology and had their domain values equal to the *SpatialEntity* class and avoided us from redefining similar properties in the Confoaf ontology.

We added the *SoftwareAgent* class as well as the property '*preferences*' that has our software ontology as its range in order to describe the preferences of this software agent class. We also added the property *hasDevice* that relates an agent, be it a software agent or a person, to a device. The *hasAgent* property defines software agents that might be associated to persons, the property *performs* relates actions to agents and finally, the *hasActivity* property that links activities to agents. We addressed the reusability requirement using the similarity principle as we did in the location ontology. The *AgentSimilarity* class achieves this requirement. It is used to state the similarity among different agents or the similarity between Confoaf profiles that belong to the same agent.

4.5.2 Example of the Actor Ontology

Figure 4-8 illustrates an example of the actor ontology that elaborates its key features. The figure is an excerpt of the actor profile that belongs to person *Khedr*. The profile includes personal contextual information such as the first and family name of that person and other persons that he knows such as person *Hamid*.

The profile also shows that the location information of this person is *MMARL*, which was defined in the example of the location ontology presented in section 4.3.3. This also demonstrates how different ontologies are composed together in our system.

In addition, the profile shows that person *khedr* has as an activity *meetingA* that was defined in the example of the time ontology in section 4.4.1 and that he is a *presenter* in that meeting. Finally, the profile shows that *Khedr* has *mkhedr* as his personal software agent, which is defined as a JADE-type agent.

```
<Person rdf:ID="Khedr">
        <firstName rdf:resource="#Mohamed"/>
        <family_name rdf:resource="#khedr"/>
        <knows rdf:resource="#Hamid"/>
        <hasAgent>
            <SoftwareAgent rdf:ID="mkhedr">
                <preferences>
                    <SWcnfg:AgentPlatForm rdf:ID="MMARL_JADE_1099"/>
                </preferences>
            </SoftwareAgent>
        </hasAgent>
        <hasActivity rdf:resource="&ConTime;meetingA"/>
        <performs>
            <ConAct:Action rdf:ID="Action">
                <ConAct:name>Presenter</ConAct:name>
            </ConAct:Action>
        </performs>
        <hasContactInfo>
            <AgentContactInfo rdf:ID="khedrContInfo">
                <workplaceHomepage>
                    <Document rdf:ID="workURL">
                        <siteURL>www.site.uottawa.ca/~mkhedr</siteURL>
                    </Document>
                </workplaceHomepage>
                <Loc:hasLocData rdf:resource="&Loc;AddressB502"/>
            </AgentContactInfo>
        </hasContactInfo>
        <hasWebId>
            <WebPresenceID rdf:ID="KhedrWebID">
                <nickname>mkhedr</nickname>
                <siteURL>http://www.yahoo.com</siteURL>
            </WebPresenceID>
        </hasWebId>
        <Loc:hasPlaceInstance rdf:resource="&Loc;MMARL"/>
</Person>
```

Figure 4-8 Example of actor ontology

4.6 The Device Ontology

The previous three sections described ontologies belonging to the *ContextFeature* class. These ontologies represented the location, time, and identity of the Go5 context that we discussed in Chapter 2. The remainder of the Go5 contexts, i.e. interest and preference, are represented in the *ContextProfile* class.

The context information, *interest*, is related to actors in the environment and is modelled by the *Activity* and *Action* profiles while the context information, *preference,* models the *Software*, *Hardware*, and *Network* profiles related to devices and services found in the environment. In this section, I will present the *Device* ontology that acts as the factory class used to describe the *preference* contexts of any resource in the environment.

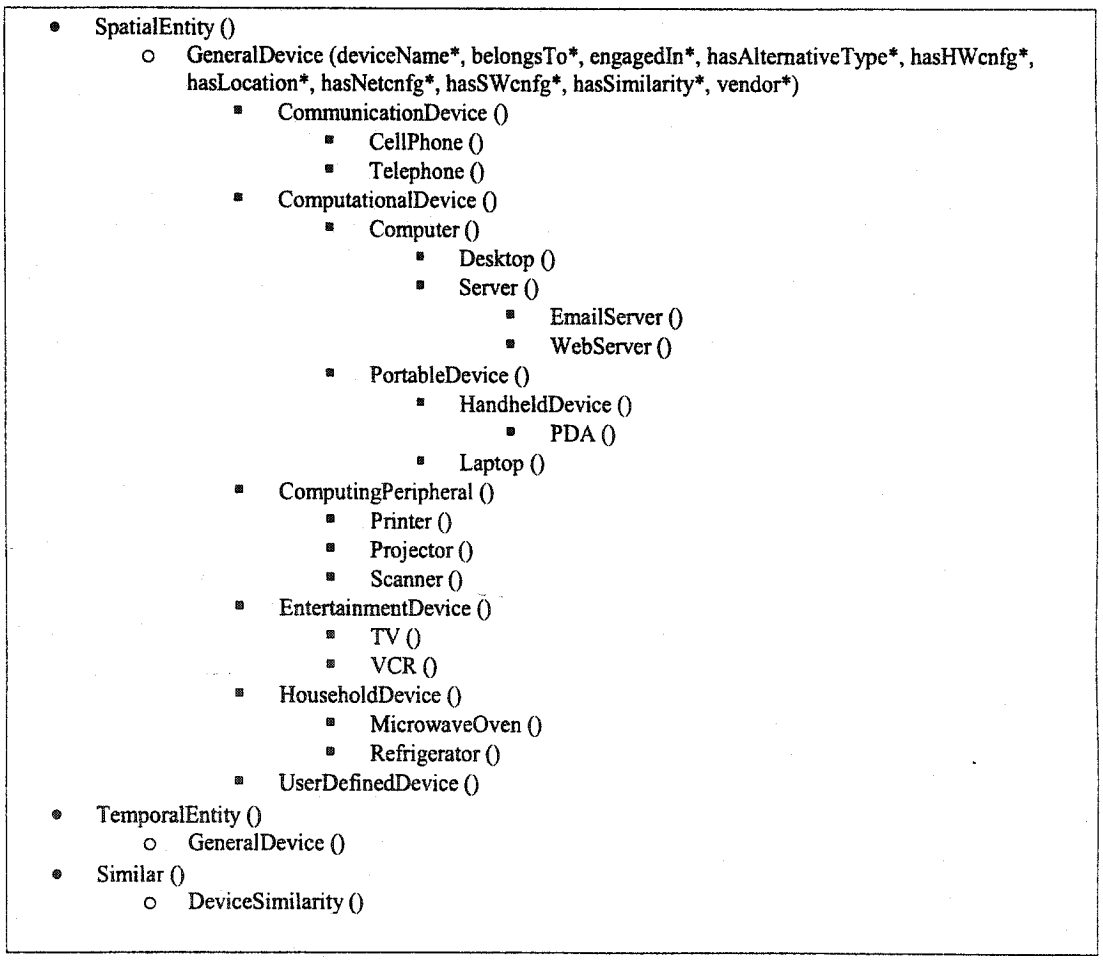


Figure 4-9 Excerpt of the device profile ontology

4.6.1 Context Information in the Device Ontology

Reusability and extensibility are the main design aspects in every type of ontology that we developed. Abstraction is the approach we took to achieve this in the device ontology and the preferences profiles. This is because abstraction provides an interface for creating families of related or dependent classes without specifying their concrete implementation.

The concept is to define an abstract class that has all the common properties found in any device expected in the environment and then instantiate from it concrete classes that define specific aspects of their associated device types.

Figure 4-9 illustrates this concept. The *GeneralDevice* is the abstract class that includes the *vendor, belongsTo*, and *deviceName* as its static preferences and *hasSWcnfg, hasHWcnfg, hasNetcnfg* as its dynamic preferences. The *hasAlternativeType* property addresses the mapping functionality between different representations of the device and the *hasSimilarity* property provides the extensibility requirement. In addition, we defined the *GeneralDevice* to be subclass of the *SpatialEntity* and the *TemporalEntity* in order to associate location and time information to the defined devices.

The device ontology has a set of concrete classes that sub-classes the *GeneralDevice* class into more domain-specific devices such as the *ComputationalDevice* class and *CommunicationalDevice* class. If a certain device has more than one domain, it will instantiate one of those domains and then uses the *DeviceSimilarity* class to associate it with any other applicable domains.

We cannot claim that our device ontology, or any of our developed ontologies, is complete but we can ensure that every ontology that we designed is perfectly applicable to the domain it is used in. This premise is verified by the extensibility and reusability features that our ontologies have.

Reusability is manifested by the abstraction method while the *hasSimilarity* property and the *UserDefinedDevice* class provide the required extensibility of the ontology. The *UserDefinedDevice* is used by application developers to represent devices that do not fall into one of the built-in device categories in the ontology. In conjunction with the

hasSimilarity property, developers can relate their custom-based devices to any other device using the fuzzy similarity approach.

In addition, reusability is attained by separating device intentional use from its preferences. The device ontology describes the intentional use while profile ontologies describe the device preferences.

4.6.2 Context-Based Preference Profiles

- HardwareCharacteristics (describesDevice*, hasOperationCharc.*)
  - VendorDefinedCharacteristics (languageUsed*, dataParameterValue*, objectParameterValue*, parameterName*)
- OperatingCharacteristics (moreInfoLink*, otherParameters*)
  - CommunicationCharacteristics (supportCommunicationType*, transmissionRate*)
  - ComponentsCharacteristics ()
    - BatteryCharacteristics (batteryLife*, batteryValue*)
    - StorageCharacteristics ()
  - InputCharacteristics ()
    - InputKeyBoard (SupportsTextType*)
    - PowerSupply (inputCurrent*, inputVolatage*, voltageType*)
  - MemoryCharacteristics ()
  - NetworkCharacteristics (hasNetworkProfile*)
  - OutputCharacteristics ()
    - DisplayCharacteristics (hasColorResolution*, hasResolution*, hasheight*, haswidth*)
    - HradCopyCharacteristics (paperType*, printingSpeed*)
  - SoftwareCharacteristics (hasSoftwareProfile*)
- Similar ()
  - HardwareSimilarity ()
- SoftwareCharacteristics (hasDataParmValue*, hasObjectParmValue*, hasParameterName*, minSystemRequirements*)
  - VendorDefinedCharacteristics ()
    - SoftwareResources ()
- SystemRequirements (hasBrowserSupport*, memoryRequirements*, requiresDeriver*, requiresHW*,usesOS*)
- AgentPlatform ()
  - instance JADE
  - instance LEAP
  - instance FipaOS
- Browser ()
  - instance InternetExplorer
  - instance NetScape
  - instance Mozilla
  - instance RealOne
- Drivers (compatibleWith*, driverName*)
- OperatingSystems ()
  - instance Linux
  - instance MacOS
  - instance RedHat
  - instance Win2000
  - instance WinCe
  - instance WinXp
- Similar ()
  - SoftwareSimilarity ()

Figure 4-10 The HW profile and SW profile ontologies

Figure 4-10 illustrates the hardware and software profiles that ACAI uses to describe semantically the preferences of devices and services in the environment. Both profiles share the similar ontological structure for describing the characteristics of these entities.

xxCharacteristics are the classes used to describe the parameters of the entities in focus, *hasParameterName* is the property that defines the name of the preference. This preference can have a data type range, such as an integer or a string, and is defined using *hasDataParmValue* property or it can have an object type range, such as a network profile or an operating preference, and is defined using *hasObjectParmValue* property.

The *OperatingCharacteristics* class defines specific preferences of an entity such as the input, output, communication, network, and software characteristics of a device.

Since we cannot anticipate every preference of every device that might exist in a pervasive environment, we defined the *xxSimilarity* and *VendorDefinedCharacteristics* classes. The *xxSimilarity* class is used to relate preferences of different devices or services together using fuzzy logic, while the *VendorDefinedCharacteristics* class is used to define additional preferences that are not defined in the current version of these profiles. This will also provide the extensibility requirement to the developed profiles in the same manner as they were used in the previously presented ontologies.

4.6.3 Example of the Preferences Profiles

Figure 4-11 is an excerpt of the contextual information that defines the preferences of a printer used in the conferencing scenario that we will discuss in the implementation chapter. The figure shows that the printing service has the following contexts and profiles:

– The printer description: The device ontology is used to define the general characteristics of the printer such as vendor information, location information, agent information, and refrences to the different profiles that define specific pereference of the printer.

– The network profile: The network information of the printer is defined using the network profile ontology. The figure shows the network charactersitcs related to the TCP/IP protocol, such as the Network standard supported by the printer, the DHCP configuration infroamtion, and Gateway infroamtion.

– The software profile: This profile describes the software requirements for running the printer. Typical preferences coded in the software profile are the operating system supported by the printer, the deriver information, and default properties of the printer.

– The hardware profile: The hardware profile ontology defines the physical preferences of the printer. As shown in Figure 4-11, typical preferences coded in this profile include memory size, processor speed, and printing speed.

Device Printer

```
<Printer rdf:ID="Lexmark_E323">
    <vendor>Lexmark</vendor>
    <!--Location information of the printer-->
    <hasLocation rdf:resource="#Office_fifth_floor_CBY"/>
    <!--user agent of the printer-->
    <belongsTo>
        <Confoaf:SoftwareAgent rdf:ID="PrinterAgent05">
            <Confoaf:preferences
        rdf:resource="&Confoaf;MMARL_JADE_1099"/>
        </Confoaf:SoftwareAgent>
    </belongsTo>
    <!--Hardware configuration profile of the printer-->
    <hasHWcnfg rdf:resource="#processor"/>
    <hasHWcnfg rdf:resource="#printingpreferences"/>
    <hasHWcnfg rdf:resource="#installedmemory"/>
    <!--Software configuration profile of the printer-->
    <hasSWcnfg rdf:resource="&SWcnfg;WinXp"/>
    <hasSWcnfg rdf:resource="#Driver"/>
    <!--Network configuration profile of the printer-->
    <hasNetcnfg rdf:resource="#TCP/IP"/>
    <!--supports other devices-->
    <hasAlternativeType rdf:resource="#LexmarkScanner"/>
    <Fuzz::hasSimilarity rdf:resource="#LexmarkE333"/>
</Printer>
```

Other Devices

```
<Printer rdf:ID="LexmarkE333"/>

<Scanner rdf:ID="LexmarkScanner"/>
```

Device Location

```
<Loc:UserDefinedPlace rdf:ID="Office_fifth_floor_CBY">
  <Loc:hasLocationRelation rdf:resource="#PRelation1"/>
  <Loc:hasUniqueID>A506</Loc:hasUniqueID>
  <Loc:hasLocationRole>
    <Loc:Educational rdf:ID="GraduateOffice"/>
  </Loc:hasLocationRole>
  <Loc:hasLocationDataType rdf:resource="&Loc;MmarlGPS"/>
</Loc:UserDefinedPlace>
```

Physical Relation

```
<Loc:PhysicalRelation rdf:ID="PRelation1">
    <Loc:hasObject>
        <Loc:Building rdf:ID="CBY"/>
    </Loc:hasObject>
    <Loc:hasPredicate>In</Loc:hasPredicate>
    <Loc:hasSubject rdf:resource="#Office_fifth_floor_CBY"/>
    </Loc:PhysicalRelation>
```

Network Profile

```
<NetProf:Networkcharacteristics rdf:about="#TCP/IP">
  <NetProf:supportProtocol>
    <NetProf:ProtocolCharacteristics rdf:ID="PrinterProtocol">
      <NetProf:supportStandard
     rdf:resource="&NWcnfg;Ethernet10M"/>
      <NetProf:supportStandard
     rdf:resource="&NWcnfg;Ethernet100M"/>
    </NetProf:ProtocolCharacteristics>
  </NetProf:supportProtocol>
  <NetProf:hasnetworkParameter>
    <NetProf:NetworkParameter rdf:ID="DHCP">
      <NetProf:parameterValue>Yes</NetProf:parameterValue>
      <NWcnfg:hasIP>137.122.86.233</NWcnfg:hasIP>
      <NWcnfg:hasSubnet>255.255.254.0</NWcnfg:hasSubnet>
      <NetProf:parameterName> EnableDHCP
</NetProf:parameterName>
    </NetProf:NetworkParameter>
  </NetProf:hasnetworkParameter>
  <NetProf:hasnetworkParameter>
    <NetProf:NetworkParameter rdf:ID="Gateway">
      <NWcnfg:hasIP>137.122.86.1</NWcnfg:hasIP>
    </NetProf:NetworkParameter>
  </NetProf:hasnetworkParameter>
</NetProf:Networkcharacteristics>
```

Hardware Profile

```
<HWcnfg:ComponentsCharacteristics rdf:ID="processor">
    <HWcnfg:otherParameters>
        <HWcnfg:VendorDefinedCharacteristics>
            <HWcnfg:parameterName>Processor Speed
            </HWcnfg:parameterName>
            <HWcnfg:dataParameterValue>200 MHz
            </HWcnfg:dataParameterValue>
        </HWcnfg:VendorDefinedCharacteristics>
    </HWcnfg:otherParameters>
</HWcnfg:ComponentsCharacteristics>

<HWcnfg:HradCopyCharacteristics
     rdf:ID="printingpreferences">
    <HWcnfg:printingSpeed>19ppm</HWcnfg:printingSpeed>
    <HWcnfg:paperType rdf:resource="&HWcnfg;A4"/>
    <HWcnfg:paperType rdf:resource="&HWcnfg;Letter"/>
    <HWcnfg:supportsFileType
     rdf:resource="&HWcnfg;PDF"/>
</HWcnfg:HradCopyCharacteristics>

<HWcnfg:MemoryCharacteristics rdf:ID="installedmemory">
    <HWcnfg:size>16MB</HWcnfg:size>
</HWcnfg:MemoryCharacteristics>
```

Software Profile

```
<SWcnfg:VendorDefinedCharacteristics rdf:ID="Driver">
    < SWcnfg:hasParameterName>LX E323</ SWcnfg:hasParameterName>
    < SWcnfg:hasDataValue>12.34.xdr </ SWcnfg:hasDataValue>
</SWcnfg:VendorDefinedCharacteristics>
```

Figure 4-11 Excerpt of the profiles developed for the printing service

The last contextual profile that we present addresses the activities that take place in an ACAI environment. Activities are defined in terms of what preferences they have, what situations they belong to and what actions are to be taken. The Activity, ActivityProfile and Action ontologies represent each of those terms successively.

Key classes in these ontologies are

The *ConditionalEquivalence,* which is used to state if two activities are equivalent based on the occurrence of a predefined condition using the property *coRule.* This property takes its range of values from the Go5 contexts.

The *ActivitySimilarity* class, which is used to define similarities among activities. It is at the same time a subclass of the fuzzy *Similar* class. This allows for the relaxation in the preferences used to decide that two activities are similar.

The *ActivitySituationModel* and the *ActivityRelation* classes are used to describe the situation at which an activity can be engaged and the relations that an activity has with other activities running in the environment.

The *ActivityFeatures* class represents the preferences of the activity and the situation where this activity is used. This is defined using the property *usedinSituation.* We also defined the property *hasAlternativeProfile* that addresses the extensibility of these ontologies. It is used to represent the same activity with more than one profile thus giving ACAI the opportunity to switch between profiles in case of inability to locate or use the default profile.

With this set of developed ontologies for representing context information in pervasive environments, ACAI can now interpret and model context captured from the sensing layer semantically. However, this does not mean that ACAI's agents can provide context-aware services and information to applications and users since these agents are still lacking the features of managing, exchanging, interacting, and manipulating context information. In the next chapter, we will present our approach for addressing these issues in order to make ACAI a context-aware infrastructure.

4.7 Summary

The development of context-aware applications can be enhanced through an infrastructure that provides application developers with the appropriate methods to define aspects of context that are relevant to the execution of the application and how applications and infrastructure components should adapt to context changes.

This chapter showed how the principle of ontology is used to represent semantically context information in order to fulfill these goals.

We presented the ontologies designed and developed for modeling the different aspects of context especially the Go5: location, time, identity, interests, and preferences. We demonstrated with examples the process of modeling context using these ontologies. We also showed that these ontologies not only model context but also include constructs that address the ability to extend and reuse these ontologies as well.

There are several potential challenges that still need to be addressed in order to use ontologies for modeling context. In particular, pervasive environments require new approaches for reasoning about context as well as approaches for the successful interaction between the components of the infrastructure, which should also be based on context. We present in the next chapter these aspects and the approaches developed to solve these aspects.

Chapter 5

Managing Context Information

An Infrastructure that supports context-aware applications must address two crucial objectives. First, it has to provide means for effective representation of context, and second, it has to support techniques for managing contextual information [113]. We argued in the previous chapter that our approach to represent context using ontologies granted context-aware applications with common understanding of context and provided the reusability and extensibility requirements.

Remarkably, the decision to use ontologies for representing context simplified the second objective. Clarifying what we mean by "*managing context*" explains the reason behind this premise.

Context management, in this thesis, stands for the ability of the infrastructure to gather, manipulate, disseminate, and infer right decisions. Decisions that are based on the level of "certainty" about the correctness of contextual information in order to assure that all entities in the environment know what they need to know and are able to influence others with what they know. Levels of "certainty" are acquired using inductive fuzzy logic reasoning, which when integrated with the context negotiation process will enable the infrastructure to adapt its behaviour automatically and manage context effectively in an uncertain and vague environments.

With this clarification, it is easily deduced that the semantic representation of context facilitates the manipulation of contexts as entities share this common understanding of what contexts mean. However, semantic modeling of context using ontologies lacks the appropriate method to represent the level of "certainty" in captured context. It is also

weak in providing adaptability and approximate reasoning about context information in order to manage the requirements of entities in the environments automatically.

This chapter discusses our approach to support levels of certainty in context-aware environment using inductive fuzzy reasoning over semantically modeled context. In the first section, we explain the reason for using fuzzy logic in our ACAI infrastructure. Next, we illustrate the process of fuzzifing ontology-based contexts with the developed fuzzy ontology. The last part of the chapter deals with the process of reasoning about context using the inductive fuzzy reasoning mechanism.

5.1 Extending Ontologies of Context with Fuzzy Logic

The concept of a fuzzy logic was introduced by Zadeh [120],[121] to deal with the representation of classes whose boundaries are ill-defined, or flexible. It does so by means of characteristic functions that take values from the interval [0,1]. A fuzzy set F in referential U is thus characterized by a membership function μ_F: $U \rightarrow [0, 1]$. The value $\mu_F(u)$ represents the "grade of membership" of u in F. Specifically, $\mu_F(u) = 1$ reflects full membership of u in F, while $\mu_F(u) = 0$ expresses absolute non-membership in F.

Classical sets can be viewed as special case of fuzzy sets, where only full membership and absolute non-membership are allowed (they are called crisp sets, or Boolean sets).

Based on this definition of fuzzy logic, extending ontology-based context information will have the following general model:

$$F = \{(x, \mu_F(x)) \mid x \in X\}$$

Where

- F is a fuzzy set in X.
- X is universe of discourse of the context variable (constructs in the ontology)
- x is the allowed values for the context variable. i.e, its ontology range
- $\mu_F(x)$ = grade of memebership function assoicated with context variable.

This model applies to any type of contexts that we discussed in Chapter 4. For example, suppose that by using the actor and location ontology, I can define the following context: *locatedIn*(khedr, B502). This crisp semantic representation states "khedr is located in

B502" and with default certainty value equal to one. If the system is uncertain about the location of the actor, it can represent this uncertainty using the following fuzzy-based information.

$$locatedIn(khedr, B502) = F = 0.1/B505 + 0.4/B504 + 0.8/B502 + 0.4/B501$$

where

F is a discrete fuzzy set

X is universe of discourse of the context variable locatedIn.

x allowed values = {B505, B504, B502, B501}.

$\mu_F(x) = (0.1, 0.4, 0.8, 0.4)$ Example1

This fuzzy-based, semantically modeled, context information means that khedr can be located in B505 with certainty value equal to 0.1 or he can be located in B504 with certainty value 0.4, or he can be located in B502 with certainty value equal to 0.8, or he can be located in B501 with certainty value equal to 0.4.

Noticeable observations are that the system is no longer as certain as it was in the crisp case, and its certainty that khedr is located in B504 is the same as that of B501. This enriches ACAI with the ability to reason about uncertain context since it can now take alternative decisions for actions that rely on the context of the user, and can cope with the uncertainty in this context information.

Three steps are required for representing uncertain contexts. The first step is extending the ontologies, which represent crisp contextual information, with the adequate properties to incorporate fuzzy logic. The second step is designing ontology for representing the vocabularies used in fuzzy logic and approximate reasoning. The third step is developing the fuzzy inference mechanism needed to reason about context.

The first step is already discussed in Chapter 4 with the built-in classes of similarity and the property *hasSimilarity* that has the *Similar* class defined in the fuzzy ontology as its range and the property *hasMF*, i.e. has fuzzy membership function, that relates context classes to the fuzzy ontology.

For the second step, we modeled common fuzzy terms and vocabularies in the form of OWL ontology[2] that allows other ontologies to import and to subclass concepts defined in this fuzzy ontology.

5.2 The Fuzzy Ontology

The fuzzy ontology defines the common concepts and terms used in fuzzy logic theory. It also defines appropriate constructs for representing fuzzy rules. These rules can be used by entities in ACAI to set conditions or trigger actions. For example, the CMA can use the following fuzzy rule to set the condition required for granting services to users in a meeting.

> **If** *UserLocation is SmallerOrEqual to MeetingRoomA_Proximity with Gaussian MF that has mean=10, variance=5* **Then** *ServiceGrant is PrinterGrant with a crisp MF equal true*.....Rule_1

The fuzzy ontology, shown in Figure 5-1, has four key classes:

5.2.1 The Fuzzy Membership Function Class

The *FuzzyMembershipFn* class is used to transform any crisp context to a fuzzy term that takes its graded values from some characteristic function. This function could have continuous or discrete values. The *Trapezoidal, Triangular, Gaussian, Sigmoid,* and *UserDefined* classes are types of the continuous functions defined in the ontology. They are also arranged in the form of an "*is a*" relationship, for example *Triangular is a Trapezoidal,* which gives the inference agent the ability to reason with alternatives in case the reasoning results are inadequate. For example in Rule_1 example, the inference agent can use the *Sigmoid* function instead of the *Gaussian* in order to have a wider range in the input location values and thus can extend this service granting rule to more users.

The discrete function is a special case of the fuzzy membership functions where the input values are finite and specific. It is used with context information that is disconnected in

---

[2] http://www.schemaweb.info/schema/SchemaDetails.aspx?id=159

nature like "number of copies" in the printer preferences, or "rooms" in a certain floor as was illustrated in example 1.

- FuzzyRule (hasAntecedent*, hasConsequent*, hasID*, hasInferenceType*)
  - Complement ()
  - Tconorm ()
    - Max ()
    - Sum ()
  - Tnorm ()
    - Min ()
    - Product ()
- Consequent (hasOperationType*)
- Antecedent (hasOperationType*)
- ConflictResolution ()
  - instance MostFrequent
  - instance Periority
  - instance RecentlyAdded
- FuzzyInference (hasResolutionMethod*)
  - Inductivefuzzy (hasIFT_URL*)
    - Node (isaLeaf*)
      - Child (isaLeaf)
      - parent (hasNodes*, isaLeaf[0])
    - Tree (hasHeuristicType*)
      - Branch ()
  - MinMax ()
    - Conditional ()
      - Unqualified ()
      - Qualified (hasHedge*)
- FuzzyMembershipFn (hasAssociatedFuzzyTerm*, operatedUpon*)
  - Continous (hasEndPoint*, hasStartPoint*)
    - Gaussian (hasVariance*, hasmean*)
      - Sigmoid ()
    - UserDefined ()
    - Trapezoidal (hasFirstpnt*, hasScndpnt*)
      - Triangular (hasCenter*)
  - Crisp ()
  - Discrete (hasInputValue ,hasMembershipValue)
- FuzzyNumber (hasValue*)
- FuzzyOperation ()
  - Defuzzification (hasDefuzzificationType*)
  - Fuzzification (hasFuzzificationType*)
  - Hedges (hasHedgeFormula*, hasHedgeName*)
  - Operation ()
    - instance And
    - instance Bigger
    - instance BiggerOrEqual
    - instance Equal
    - instance Or
    - instance Smaller
    - instance SmallerOrEqual
- FuzzySet (AggregationOf*, alphacutSupport*, hasCardinality*, hasMembershipFn*)
  - FuzzyTerm ()
- Similar (degreeOfCertain*, similarIn*, similarTo*)

Figure 5-1 Class-Property hierarchy diagram of the Fuzzy ontology

UserDefined membership function provides the extensibility requirement to this ontology. A developer uses this class if his context information cannot be fuzzified with one of the defined membership functions. In this case, the developer provides the inference agent with a set of $(x, \mu_F(x))$ describing his context information. Consequently, the inference agent applies the Lagrange interpolation method on this set of values to provide a continuous, user-defined, fuzzy membership function.

5.2.2 The Fuzzy Rule Class

The *FuzzyRule* class defines rules in the form of *If Antecedents (operand used) with MFs then consequents with MFs,* where *Antecedent* and *consequent* has the form of (variable, operand, value).

Variables and values have *owl:Thing* as their range, which means that they can be linked to any external instance defined in the context ontologies we designed and explained in Chapter 4.

For example, in the event-notification application that we will discuss in Chapter 6, the inference agent uses the following rule to decide the level of alert to send to the users. This alert is based on the users' location and the time of the activity.

```
<Fuzz:Min rdf:ID="TimeRule">
 <Fuzz:hasAntecedent>
  <Fuzz:Antecedent rdf:ID="TimeAnt2">
   <Fuzz:hasOperationType rdf:resource="#Equal"/>
   <Fuzz:hasFuzzyValue rdf:resource="#Far"/>
   <Fuzz:hasFuzzyVariable>
    <Fuzz:FuzzyVariable rdf:ID="UserLocation"/>
   </Fuzz:hasFuzzyVariable>
  </Fuzz:Antecedent>
 </Fuzz:hasAntecedent>
 <Fuzz:hasAntecedent>
  <Fuzz:Antecedent rdf:ID="TimeAntc1">
   <Fuzz:hasOperationType>
    <Fuzz:Operation rdf:ID="at"/>
   </Fuzz:hasOperationType>
   <Fuzz:hasFuzzyVariable rdf:resource="#TimeVariable"/>
   <Fuzz:hasFuzzyValue rdf:resource="#meetingtime"/>
  </Fuzz:Antecedent>
 </Fuzz:hasAntecedent>
 <Fuzz:hasConsequent rdf:resource="#TimeAlert"/>
</Fuzz:Min>
```

```
<StandardTime rdf:ID="meetingtime">
  <hasTimevalue>2004-01-27T12:05:00
</hasTimevalue>
 </StandardTime>

 <Fuzz:Consequent rdf:ID="TimeAlert">
  <Fuzz:hasFuzzyVariable>
   <Fuzz:FuzzyVariable rdf:ID="alert"/>
  </Fuzz:hasFuzzyVariable>
  <Fuzz:hasFuzzyValue>
   <Fuzz:FuzzyTerm rdf:ID="High"/>
  </Fuzz:hasFuzzyValue>
 </Fuzz:Consequent>

 <Fuzz:FuzzyVariable rdf:ID="TimeVariable">
  <Fuzz:pointsTo>
   <TemporalEntity rdf:ID="meetingA"/>
  </Fuzz:pointsTo>
 </Fuzz:FuzzyVariable>
```

The Rule states: If (UserLocation, Equal, Far) and (meetingA, at, meetingtime) then (Alert, Equals, High). The rule also illustrates how ontologies are combined together. This is shown in the temporal entity "*meetingA*" which we defined using the time ontology, and is fuzzified with the fuzzy variable "TimeVariable".

5.2.3 The Similar Class

The *Similar* class is the link used by other ontologies to describe the similarity between concepts. The *Similar* class has three properties for defining fuzzy similarity:

- similarTo property: is used to related the original concept with another concept. For example a location instance could have the following triple (B502, similarTo, MMARL) which states that B502 is similar to MMARL or an actor instance could have the triple (Khedr, similarTo, mkhedr) which means that Khedr is similar to mkhedr.
- similarIn property: is used to state what these concepts are similar in. For example, (B502, similarIn, Room) which means that B502 is similar to MMARL and they are similar in their type, being a room. Also, (khedr, similarIn, PhDStudent) means that khedr and mkhedr are similar in their role, being a PhD student.
- degreeOfCertain property: is used to describe how accurate the similarity is between the concepts. For example, (B502, degreeOfCertain, 0.8) means that B502 is similar to MMARL with 80% degree of certainty.

5.2.4 The Fuzzy Inference Class

Many methods were developed for reasoning using fuzzy logic [118] and each has its approach of deducing knowledge from existing facts. However, no single inference method is adequate for all situations. Hence, we semantically represented these methods in the fuzzy ontology in order for the inference agent to execute the appropriate method based on context. In particular, the fuzzy ontology defines the approximate and inductive reasoning mechanisms since they are the two types of inference used in this thesis.

However, other developers can benefit from the extensibility feature in our ontology to define their own inference process.

The approximate reasoning class addresses the conditional-unqualified and conditional-qualified methods.

In the first case, rules are expressed in the form p : *If* χ is A, *then* y is B, where χ,y are the context variables and A,B are the fuzzy sets that represent the universe of discourse (X, Y) of these contexts respectively. This type of inference can be viewed in the form of fuzzy relation $R(\chi,y)= \wp[A(x), B(y)]$ and $\wp(a,b)=\min(1,1-a+b)$ where R is a fuzzy set on the cartesian product X•Y that is calculated over each $x \in X$ and $y \in Y$ using the fuzzy implication operation $\wp$.

Thus, for a rule like Rule_1, we can say that A= "*SmallerOrEqual to MeetingRoomA_Proximity*" while B= "*PrinterGrant*". If we assume that A has the MF of$\{0.1/x_1 +0.8/x_2 +1/x_3\}$ and B has MF $\{0.5/y_1 +1/y_2\}$ then $\wp(a,b)$ will be equal to $\{1/x_1,y_1 +1/x_1,y_2 +0.7/x_2,y_1 +1/x_2,y_2 +0.5/x_3,y_1 +1/x_3,y_2\}$. This means that if a user has the location x_2 then ACAI can grant him access to printer y_1 with a degree of certainty, T(p), equal 0.7 or to printer y_2 with T(p)=1. After using the approximate reasoning, the inference agent can successfully grant the user access to printer y_2 as it has a higher membership value.

The conditional-qualified method has the form of p : *If* χ is A, *then* y is B is S and applies the same implication process as in the conditional-unqualified method except for the last step where the output is passed through a hedging filter, *S*, in order to increase or decrease the degree of certainty of the outcome.

The inference agent in ACAI uses the approximate reasoning when the number of context variables is relatively small. This is because the number of rules increases exponentially with the number of variables used in this type of fuzzy inference method.

In situations where the number of context variables is large, the inference agent will use the inductive reasoning. This is because the inductive reasoning can detect patterns in the input values, and remove irrelevant contexts, which will have the result of reducing the number of generated rules and enhance system dynamism.

In both cases, the inference agent must perform the following steps:

1. Define linguistic terms and generate membership functions.
2. Build the fuzzy inductive inference tree (case of inductive reasoning only).
3. Generate Rules from fuzzified context information.
4. Perform the inference process.

Figure 5-2 illustrates these steps and depicts the process of binding fuzzy logic with the context information described using ontologies.

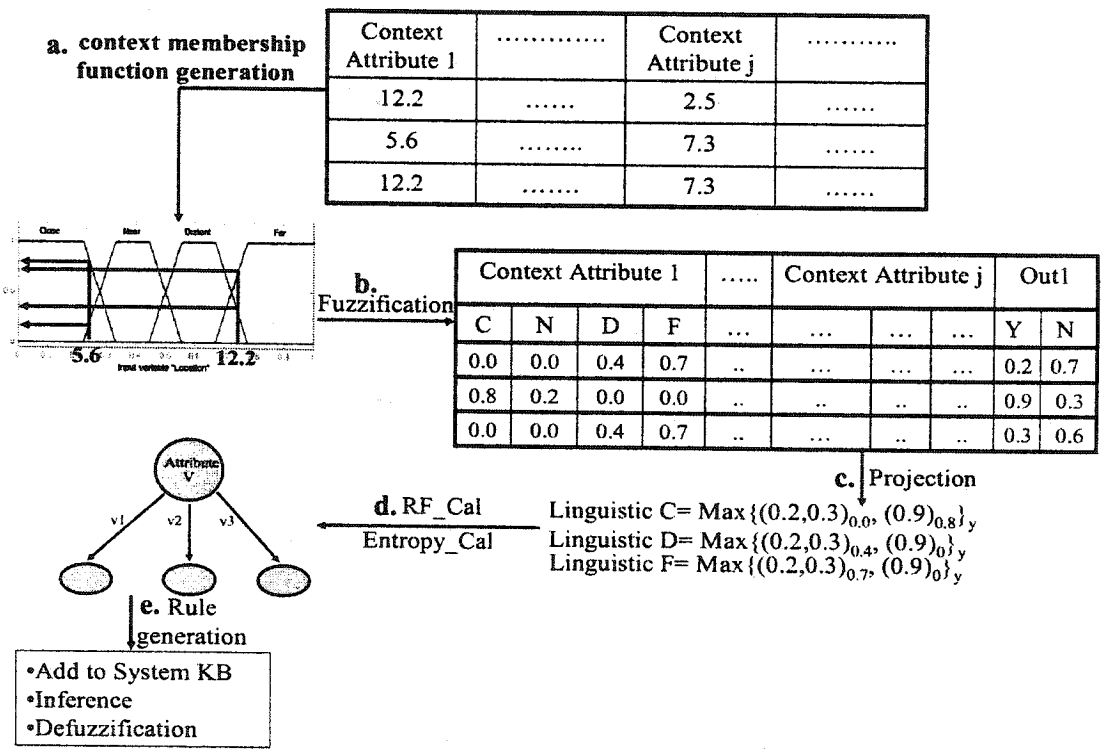


Figure 5-2 Steps used in the fuzzy inference of context information.

5.3 The Fuzzy Inference of Context

Fuzzy inference works on input values that have the form of $\sum_{i=1}^{N} \mu(x_i)/x_i \mid x \in X$ in case of discrete input or the form of $\int_X \mu_F(x)/x$ in case the input is continuous. This means that ACAI's inference agent needs first to loosen the semantically modeled context before starting the inference.

Context fuzzification (steps "a" and "b" in Figure 5-2) means transforming the ontology-based context instances into a set of linguistic terms. Each linguistic term has a certain membership function that is used in the inference instead of the original context instance as in the case of standard logic.

A relation between a device and its owner is an example that illustrates the fuzzification of context. A crisp relation will have the following ontological representation

The captured context indicates that *MMARL-10* is a device of type *Desktop*, it is located in *B502*, its vendor is *IBM* and it belongs to user *Khedr*.

```
<Desktop rdf:ID="MMARL-10">
 <hasLocation rdf:resource="&Loc;B502"/>
 <belongsTo rdf:resource="&Confoaf;Khedr"/>
 <vendor>IBM</vendor>
 </Desktop>
```

This information is acceptable if user *Khedr* is actually the only person who is using this desktop all the time. However, he might be using it only during the morning hours, which will make this fact inaccurate and might cause unacceptable consequences. When fuzzy logic is used in this situation to represent the uncertainty in the relation between the desktop and the user, the semantic representation will include the property *hasMF* that will link the crisp context ontology to the fuzzy ontology; then linguistic terms are defined that describe the membership functions used in the relation.

```
 <DeskTop rdf:ID="MMARL-10">
  <hasLocation rdf:resource="&Loc;B502"/>
  <belongsTo
 rdf:resource="&Confoaf;Khedr"/>
  <vendor>IBM</vendor>
  <hasMF>
    ....
```

```
<Fuzz:hasFuzzyTerm >
<Fuzz:FuzzyTerm rdf:ID="Morning">
 <Fuzz:hasMembershipFn>
  <Fuzz:trapezoidal>
   <Fuzz:hasFirstpnt>9.0</Fuzz:hasFirstpnt>
   <Fuzz:hasEndPoint>12.0</Fuzz:hasEndPoint>
   <Fuzz:hasScndpnt>11.0</Fuzz:hasScndpnt>
   <Fuzz:hasStartPoint>8.0</Fuzz:hasStartPoint>
  </Fuzz:trapezoidal>
 </Fuzz:hasMemebrshipFn>
 </Fuzz:FuzzyTerm>
```

```
<Fuzz:hasFuzzyTerm>
 <Fuzz:FuzzyTerm rdf:ID="Afternoon">
  <Fuzz:hasMembershipFn>
   <Fuzz:triangular>
    <Fuzz:hasCenter>13.0</Fuzz:hasCenter>
    <Fuzz:hasEndPoint>15.0</Fuzz:hasEndPoint>
    <Fuzz:hasStartPoint>11.0</Fuzz:hasStartPoint>
   </Fuzz:triangular>
  </Fuzz:hasMembershipFn>
 </Fuzz:FuzzyTerm>
</Fuzz:hasFuzzyTerm>
```

```
<Fuzz:hasFuzzyTerm>
    <Fuzz:FuzzyTerm rdf:ID="Night">
     <Fuzz:hasMembershipFn>
      <Fuzz:Crisp rdf:ID="Zero"/>
     </Fuzz:hasMembershipFn>
    </Fuzz:FuzzyTerm>
   </Fuzz:hasFuzzyTerm>
```

In this example, we defined three linguistic terms that describe the temporal relation between the desktop and the user. The first term is "*Morning*" that defines the degree of certainty that *Khedr* is using *MMARL-10* during the morning hours from 8:00 am to 12:00 pm. The MF is represented as a trapezoidal function indicating that between 9:00 and 11:00, ACAI is certain that *MMARL-10* belongs to user *Khedr*. The second terms is "*Afternoon*" that is described as a triangular function between the hours 11:00 am to 3:00 pm and the last term is *"Night"* which is described as a crisp membership function with value equal to zero. This means that *MMARL-10* does not belong to *Khedr* during the night hours.

With the aforementioned example that clarifies our intention of using fuzzy and fuzzy ontology to represent uncertainty in the context information, the next sections describe the steps performed by the inference agent in ACAI for reasoning about uncertain context information.

5.3.1 Defining Linguistic Terms and Generating Membership Functions.

Similar to the notions of variables and ranges of variables that exist in traditional mathematical theory, fuzzy set theory has the notion of linguistic terms which represent the universe of discourse of a linguistic variable.

Zadeh defined linguistic variables as a quintuple in the form of (ω, T(ω), X, G, M); where

- ω – the name of the linguistic variable
- T(ω) – the term set or set of fuzzy reference sets of ω
- X – the universe of discourse on which all member sets of T(ω) are defined
- G – the is a grammar comprising a set of *syntactic rules* to generate the terms in T(ω)
- M – the semantic rule for associating with each value of ω its meaning

Applying the quintuple to the *"MMARL-10 belongsTo Khedr in the Morning hours"* example, ω will represent "*MMARL-10 belongsTo Khedr*", T(ω) will represent {Morning, Afternoon, Night}, X is the time range from [8:00,20:00], G is the same as T(ω) and M will be the membership functions {trapezoidal, triangular and crisp}describing each term in the T(ω).

The critical part in the quintuple is to decide the term set, T(ω) and the membership functions, M, in such a way that will provide good modeling to the uncertainty in context and with low overhead in computation.

The inference agent is responsible for performing this task, as users and application developers are assumed not to be expert in fuzzy logic and domain mapping techniques. Application developers will only specify the number of linguistic terms to be used by the inference agent to map the context information to them.

For example, a developer may state that the location context inside a meeting room is specified by two linguistic terms "*near*" and "*far*" with respect to sensor positions in the room. These two linguistic terms, T(ω), need to be mapped to the standard membership functions defined in the fuzzy ontology and stored in the system KB. For example, the two linguistic terms can be represented by Gaussian functions that have mean and variance determined by the range of the location and the overlap region.

Pseudocode 5-1 illustrates the sequence used for fuzzifing context and elicitating membership functions representing this fuzzification of context.

To generate the required membership functions, the IA first checks the context ontology to determine the range and domain of the context construct to be fuzzified. Then the IA uses a simple algorithm to generate the membership functions, which requires least computational overhead.

The algorithm starts by dividing the context range into 2n+1 equal slots, where n is the number of linguistic terms required (in the location example above, n=2). It then assigns three slots to each linguistic term and each set of adjacent terms share one slot in common that will represent the overlapping region in the fuzzy sets. The three slots concept is chosen because the trapezoidal function, that is considered the general membership function from which other membership functions can be derived, needs four points to be defined. A Triangular, Gaussian, or S-function can easily be assigned to these regions by executing a transformation function that maps the generated trapezoidal function to any of these functions.

Inputs: *T(ω), C where:*
T(ω) is the set of linguistic terms representing a context instance.
C is the context instance

Procedure:

- ***Context Preprocessing***
  1- *Fetch the Range associated with the context C → Query the Ontology Agent.*
  2- *Provide Result to the Inference Agent.*
- ***Range Clustering*** *→ Inference Agent*
  1- *Divide the range into 2n+1 equal slots.*
  n is the number of linguistic terms required
  2- *Assigns three slots to each linguistic term.*
  3- *Each adjacent terms share one slot in common that will represent the overlapping region in the fuzzy sets*
- ***Membership function Approximation*** *→ Context Management Agent*
  1- *Sample each of the 2n+1 regions that represent the range of the context C.*
  2- *Randomly choose samples from each of these regions.*
  3- *Project these samples to the user for setting expected membership values for them.*
  4- *Provide these values to the Inference Agent.*
- ***Membership function Elicitation*** *→ Inference Agent*
  1- *Validate the received data*
  2- *If Consistent*
  a. *Apply the interpolation function on the received values.*
  b. *Decide the corresponding standard membership function using the similarity function*
  3- *Else*
  a. *Add/Remove slots from regions that include inconsistencies → non-uniform division*
  b. *Apply the interpolation function on the received values.*
  c. *Decide the corresponding standard membership function using the similarity function*

Output: *A set of Membership functions M representing the linguistic terms in the T(ω)*

Pseudocode 5-1 Generating membership functions for context information

After successfully dividing the context range into its 2n+1 regions, the CMA randomly selects samples from each region and requests from the application developer, or user, to specify the expected membership value for each of these samples. This step is required in order for the inference agent to deduce correctly the best membership functions that resemble user expectation. This step also allows the inference agent to decide if the tentative membership functions should remain equally divided as initially proposed, or if they should be re-calculated and as a result become non-uniformly spaced.

After this decision, the inference agent applies the Lagrange interpolation mechanism on these sampled data followed by Euclidian measure to decide the standard membership functions that best resemble the user-defined data.

If the context range is not equally divided, the context membership generation starts with the same procedure as stated above, but assigns a different number of slots to each

linguistic term according to the required ratio. This ratio is calculated from the sampled data taken in the *"Membership function Approximation"* step shown in Pseudocode 5-1.

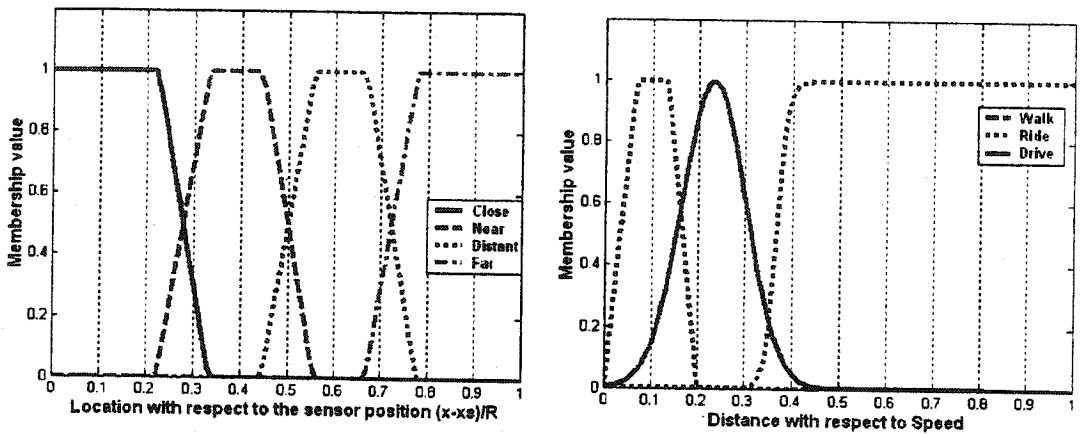


Figure 5-3 Elicitation of fuzzy membership functions

a) Uniform membership function representation of location. b) Non-uniform membership function representation of distance

Figure 5-3 demonstrates the output membership function generated from the described algorithm. The left figure represents the context information, "location", using four linguistic terms with equal regions. The right figure shows the context information, "distance", using three unequal membership functions according to a specified ratio between them 1:2:4.

5.3.2 Building the Inductive Fuzzy Tree

After deciding the membership functions for the context information captured and derived by the agent system, the inference agent builds the inductive fuzzy tree, shown as steps "c" and "d" in Figure 5-2.

In general, building an inductive fuzzy tree requires a set of training examples, a heuristic method for choosing the best attribute to split at any node of the tree, and an inference mechanism [119]. Training examples in our infrastructure are acquired either from experts in the application domain or from sensors and historical actions of users. These examples are stored in the knowledgebase that is under the control of the system knowledgebase agent and can be manipulated by application developers.

For example, in the service provisioning application that I will present in Chapter 6, context attributes such as the service capabilities are specified by the user agent. This is in contrast to context attributes such as the service availability, service and user location, and session preferences, which are provided by the context provider agents. With this information, the inference agent can proceed to build the inductive fuzzy tree.

The IA uses a heuristic method based on entropy-measurement. This method has two purposes. First, it reduces the training example-space through which the computation is performed, thus reducing complexity and increasing the speed of convergence in calculations. Second, because repeated values are averaged, it reduces the effect of a dominant linguistic term in the calculation

Inputs: *(D, A, C), where:*
D is the training set of instances
A is the context attributes used to build the inductive fuzzy tree.
C is the output context/action attribute

Output: *an inductive fuzzy tree S representing a sequential decision process for classifying instances (Predicting the values of the class attribute C); each node of S is labeled with a non-class attribute of A*

Informal Inductive Bias: minimize the average height of the tree

Procedure:

```
If the set of remaining non-class attributes is empty or if all of the instances in D are in
 the same class
        return an empty tree
else {
Compute the class entropy of each attribute over the dataset D
        let a* be an attribute with minimum class entropy
        create a root node for a tree S; label it with a*
        for each linguistic term b of attribute a* {
                let S(a*=b) be the tree computed recursively by FID on input
                 (D|a*=b, A-a*, C), where:
                D|a*=b contains all instances of D for which a* has the value b
                A-a* consists of all attributes of A except a*
                attach S(a*=b) to the root of S as a subtree
        }
        return the resulting decision tree S
```

Pseudocode 5-2 Building the fuzzy induction tree of context information

The inference agent performs the steps in Pseudocode 5-2 to build the inductive fuzzy tree. The root of the tree contains all the training examples. It represents the whole description space since no restrictions are imposed yet. Each node is recursively split by

partitioning its training examples. Training examples at the node to be split are distributed on its child nodes according to the degree with which they match the linguistic terms at the parent node. A node becomes a leaf if its associated examples come from a unique class, or when all context attributes are used along the path from the root to this node.

Specifically, the inference agent performs the following calculations in order to build the inductive fuzzy tree

- For every linguistic term T_i of context attribute A, calculate its relative frequency, p_{ij}, $p_{ij} = M(Tnorm(T_i, C_j))/M(C_j)$ with respect to each linguistic term, C_j, of the output class C. M is the cardinality measure for all projected examples.
- Calculate the total relative frequency of every linguistic term T_i of context attribute A $P_i = \sum_{j=1}^{J} p_{ij}$ where J is total number of projected examples.
- Calculate the weighted entropy $E_A = \sum w_i.P_i \quad i = 1..N$ of each context attribute A over its total linguistic terms N. The weight factor is equal to $w_i = \frac{p_i}{\sum_{l=1..N} p_l}$
- Choose the context attribute with the minimum entropy as the winning attribute.
- Classify the examples along each linguistic branch of that context attribute.
- Calculate the stopping factor, beta; if beta is bigger than certain threshold then this node is a leaf else repeat for the new node.
- For each branch in the tress, repeat the above process until all training examples are classified.

We illustrate the fuzzy induction process through a sport recommendation system that uses the small training examples shown in Table 5-1. The training examples represent the fuzzified contextual information needed to induce the type of sport that can be recommended to the users. The types of contextual information used in this application are temperature, outlook, wind, and humidity while the output is to recommend for the

users playing one of the following sports: volleyball, swimming, or weightlifting according to these contextual information.

Look Sunny	Look cloudy	Look rainy	Temp Hot	Temp mild	Temp cool	Hum Hum	Hum Normal	Wind Wind	Wind No	Volley	Swim	Weigh Lift
0	0	1	0	0	1	1	0	0.8	0.2	0	0	1
0	0.1	0.9	0.7	0.3	0	0.5	0.5	0.5	0.5	0	0	1
0	0.3	0.7	0	0	1	0	1	0.1	0.9	0	0	1
0	0.7	0.3	0	0.3	0.7	0.7	0.3	0.4	0.6	0.2	0	0.8
0	0.7	0.3	0.8	0.2	0	0.1	0.9	0.2	0.8	0.7	0.3	0.1
0	0.9	0.1	0	0.9	0.1	0.1	0.9	0.7	0.3	0	0	1
0	1	0	0	0.2	0.8	0.2	0.8	0	1	0.7	0	0.3
0.2	0.6	0.1	0.3	0.7	0	0.3	0.7	0.3	0.7	0.6	0.4	0
0.2	0.6	0.2	0	1	0	0.3	0.7	0.3	0.7	0.7	0.2	0.1
0.3	0.8	0	0.6	0.4	0	0	1	0	1	0.9	0.3	0
0.5	0.6	0	0	0.3	0.7	0	1	0.9	0.1	0	0.1	0.9
0.5	0.7	0	1	0	0	0.6	0.4	0.7	0.3	0.2	0	0.7
0.8	0.3	0	1	0	0	0.8	0.2	0.4	0.6	0.4	0.8	0
0.8	0.3	0	1	0	0	1	0	0.2	0.8	0.3	0.7	0
0.9	0.1	0	0.2	0.8	0	0.1	0.9	1	0	0	0	1
1	0	0	0.8	0.5	0	0	1	0	1	0.8	0.5	0

Table 5-1 Fuzzified context attributes and the output decision values

Figure 5-4 shows an excerpt of the semantic modeling of the sport activity using the *Activity* ontology described in Chapter 4. The figure also shows how *Temperature*, one of the contexts used in this activity, was modelled using the property *necessaryFeature* & the class *ActivityFeature.*

Each of these context terms is fuzzified to their linguistic terms using the process discussed in section 5.3.1. For example, temperature is fuzzified into three linguistic terms {Hot, Mild, Cool} with membership functions {Gaussian, Trapezoidal, Sigmoid} respectively.

```
<Contivity:Activity rdf:ID="Sports">
    <rdfs:comment>illustrates a sample of inductive fuzzy</rdfs:comment>
    <Contivity:describesedBy>
        <ActivityProfile rdf:ID="SportProfile">
            <necessaryFeatures>
                <ActivityFeatures rdf:ID="Tempereture_H">
                    <featureValue>
                        <Fuzz:Gaussian rdf:ID="GaussianHot">
                            <Fuzz:hasmean>27.0</Fuzz:hasmean>
                            <Fuzz:hasVariance>5.0</Fuzz:hasVariance>
                        </Fuzz:Gaussian>
                    </featureValue>
                    <featureName>Hot</featureName>
                </ActivityFeatures>
            </necessaryFeatures>
            <withActivityType rdf:resource="# Volley"/>
            <withActivityType rdf:resource="#WeightLifting"/>
            <withActivityType rdf:resource="#Swimming"/>
            <necessaryFeatures>
                <ActivityFeatures rdf:ID="Temperature_C">
                    <featureValue rdf:resource="#Sigm_C"/>
                    <featureName>Cool</featureName>
                </ActivityFeatures>
            </necessaryFeatures>
            <necessaryFeatures>
                <ActivityFeatures rdf:ID="Tempereture_M">
                    <featureName>Medium</featureName>
```

```
                    <featureValue>
                        <Fuzz:trapezoidal rdf:ID="Trap_M">
                            <Fuzz:hasEndPoint>25.0</Fuzz:hasEndPoint>
                            <Fuzz:hasFirstpnt>18.0</Fuzz:hasFirstpnt>
                            <Fuzz:hasScndpnt>22.0</Fuzz:hasScndpnt>
                            <Fuzz:hasStartPoint>15.0</Fuzz:hasStartPoint>
                        </Fuzz:trapezoidal>
                    </featureValue>
                </ActivityFeatures>
            </necessaryFeatures>
            <Contivity:describes rdf:resource="#Sports"/>
        </ActivityProfile>
    </Contivity:describesedBy>
</Contivity:Activity>
<Fuzz:Sigmoid rdf:ID="Sigm_C">
    <Fuzz:hasStartPoint>10.0</Fuzz:hasStartPoint>
    <Fuzz:hasEndPoint>15.0</Fuzz:hasEndPoint>
</Fuzz:Sigmoid>
```

Figure 5-4 The fuzzified semantic representation of context.

Every row in Table 5-1 represents a situation where different values were taken for the four contexts, and the type of sport that is adequate with these context values.

After the inference agent performs the entropy calculation on these examples as shown in Table 5-2, it decides that the outlook attribute is the most appropriate context to be used as the root of the inductive fuzzy tree since it has the minimum entropy

The inference agent continues the process of splitting the context attributes and categorizing the training examples until all the training examples are distributed on the nodes of the generated tree.

	Look Sunny	Look cloudy	Look Rainy	Volley	Swim	W.L	M(T1, U1,S)	M(T1, U2,S)	M(T1, U3,S)	M(T2, U1,S)	M(T2, U2,S)	M(T2, U3,S)	M(T3, U1,S)	M(T3, U2,S)	M(T3, U3,S)
	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
	0	0.1	0.9	0	0	1	0	0	0	0	0	0.1	0	0	0.9
	0	0.3	0.7	0	0	1	0	0	0	0	0	0.3	0	0	0.7
	0	0.7	0.3	0.7	0.3	0.8	0	0	0	0.7	0.3	0.7	0.3	0.3	0.3
	0	0.9	0.1	0	0	1	0	0	0	0	0	0.9	0	0	0.1
	0	1	0	0.7	0	0.3	0	0	0	0.7	0	0.3	0	0	0
	0.2	0.6	0.1	0.6	0.4	0	0.2	0.2	0	0.6	0.4	0	0.1	0.1	0
	0.2	0.6	0.2	0.7	0.2	0.1	0.2	0.2	0.1	0.6	0.2	0.2	0.2	0.2	0.1
	0.3	0.8	0	0.9	0.3	0	0.3	0.3	0	0.8	0.3	0	0	0	0
	0.5	0.6	0	0	0.1	0.9	0	0	0	0	0.1	0.6	0	0	0
	0.5	0.7	0	0.2	0	0.7	0.2	0	0.5	0.2	0	0.7	0	0	0
	0.8	0.3	0	0.4	0.8	0	0.4	0.8	0	0.3	0.3	0	0	0	0
	0.9	0.1	0	0	0	1	0	0	0.9	0	0	0.1	0	0	0
	1	0	0	0.8	0.5	0	0.8	0.5	0	0	0	0	0	0	0
M(T,S)	4.4	6.7	3.3				2.1	2	1.5	3.9	1.6	3.9	0.6	0.6	3.1
P_{ii}							0.477	0.454	0.34	0.582	0.238	0.582	0.181	0.181	0.93
Entropy(i)	1.548	1.401	0.99												
Weight	0.305	0.465	0.229												
Total entropy of this attribute			1.35												

Table 5-2 Excerpt of the process of building the induction tree

Figure 5-5 shows the inductive fuzzy tree formed using the training examples in Table 5-1 and the algorithm depicted in Pseudocode 5-2.

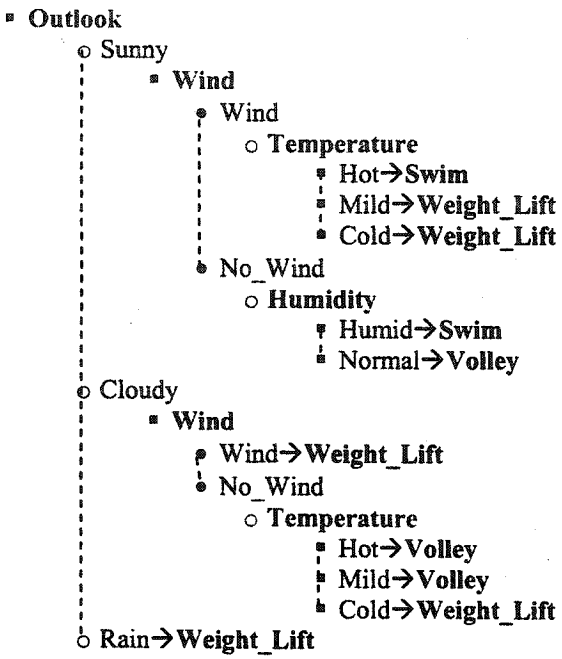


Figure 5-5 The inductive fuzzy tree

Several observations can be recorded from the generated tree. First, outlook was the most important context for this type of application, followed by wind and then temperature contexts. Second, although many factors may influence the output decision, the IA found that with rainy conditions the pattern is always towards indoor sports. Third, humidity was an important context in case of sunny outlook while it was not in case of a cloudy outlook since the impact of humidity vanishes with cooling conditions. Finally, the inference agent from this tree can generate the rules that are used in the inference process.

5.3.3 Rule Generation and Inference Process

Each path along the generated tree from root to leaf is converted into a rule with the antecedent part representing the context attributes of each succeeding branch, and the consequent part representing the class at the leaf with highest membership value. The membership value of the every output at each leaf is taken to be equal to the maximum membership value of all instances reaching that leaf and having that class as their output.

For instance, the tree in Figure 5-5 will generate the following rules:

> **RULE 0:** IF (Look is Look_Sunny) And (Wind is Wind_Wind) And (Temp is Temp_Hot) THEN (OutPut is Sport_Swim) with MF value [0.211, 0.421, 0.368]
>
> **RULE 1:** IF (Look is Look_Sunny) And (Wind is Wind_Wind) And (Temp is Temp_Mild) THEN (OutPut is Sport_Lift) with MF value [0.0, 0.0, 1.0]

RULE 2: IF (Look is Look_Sunny) And (Wind is Wind_Wind) And (Temp is Temp_Cool) THEN (OutPut is Sport_Lift) with MF value [0.0, 0.0, 1.0]

RULE 3: IF (Look is Look_Sunny) And (Wind is Wind_No) And (Hum is Hum_Hum) THEN (OutPut is Sport_Swim) with MF value [0.375, 0.625, 0.0]

RULE 4: IF (Look is Look_Sunny) And (Wind is Wind_No) And (Hum is Hum_Norm) THEN (OutPut is Sport_Volley) with MF value [0.615, 0.385, 0.0]

RULE 5: IF (Look is Look_Cloudy) And (Wind is Wind_Wind) THEN (OutPut is Sport_Lift) with MF value [0.0, 0.0, 1.0]

RULE 6: IF (Look is Look_Cloudy) And (Wind is Wind_No) And (Temp is Temp_Hot) THEN (OutPut is Sport_Volley) with MF value [0.75, 0.083, 0.167]

RULE 7: IF (Look is Look_Cloudy) And (Wind is Wind_No) And (Temp is Temp_Mild) THEN (OutPut is Sport_Volley) with MF value [0.846, 0.154, 0.0]

RULE 8: IF (Look is Look_Cloudy) And (Wind is Wind_No) And (Temp is Temp_Cool) THEN (OutPut is Sport_Lift) with MF value [0.467, 0.0, 0.533]

RULE 9: IF (Look is Look_Rainy) THEN (OutPut is Sport_Lift) with MF value [0.0, 0.0, 1.0]

After generating the fuzzy rule set from the tree, the inference agent sends the rule set to the system KB agent in order to be stored, manipulated, and queried by other agents such as the CMA and CPA.

The process of inferring new context instances is carried out from the generated rules using the compositional rule of inference method [118]. First, context instances are fuzzified by comparing their captured values with their linguistic terms. Then for each rule, the minimum value between all the fuzzified contexts in its antecedent part is selected and used as the membership value of the rule's consequent part. The outcome of these steps will be one of the following:

1- All rules have the same output consequent. In this case, the IA takes this consequent as the result of the fuzzy inference and its membership value equals to the minimum membership value of all the rules output

2- Rules have different output consequents and with different membership values. In this case, the IA takes the maximum value of all similar consequents and then selects the output consequent with the highest membership value

For example, using the generated tree shown in Figure 5-5, let a new fuzzified context instance for recommending a sport to users based on outlook, temperature, wind and humidity be equal to {Outlook (0.6, 0.4,0.1), Temperature (0.8, 0.2, 0.0), Humidity (0.4, 0.6), Wind (0.2,0.9)}. Then the output of the generated rules will be equal to

Rule 1: min(0.6,0.2,0.8)= 0.2 → Swim with MF of 0.2.

Rule 2: min(0.6,0.2,0.2)= 0.2 → Lift with MF of 0.2.

Rule 3: min(0.6,0.2,0.0)= 0.0→ Lift with MF of 0.0

Rule 4: min(0.6,0.9,0.4)= 0.4 → Swim with MF of 0.4.

Rule 5: min(0.6,0.9,0.6)= 0.6 → Volley with MF of 0.6.

Rule 6: min(0.4,0.2)= 0.2→ Lift with MF of 0.2

Rule 7: min(0.4,0.9,0.8)= 0.4→ Volley with MF of 0.4

Rule 8: min(0.4,0.9,0.2)= 0.2 → Volley with MF of 0.2.

Rule 9: min(0.4,0.9,0.0)= 0.0 → Lift with MF of 0.0.

Rule 10: min(0.0)= 0.0 → Lift with MF of 0.0.

Since there is more than one consequent that has the same output class, the IA will take the maximum value of each group of classes that have the same consequent, then the maximum of the resultant as follows

$\max(\max(0.2,0.4)_{swim}, \max(0.6,0.4,0.2)_{Volley}, \max(0.0,0.0,0.2,0.0,0.2)_{Lift} = \max(0.4_{Swim}, 0.6_{Volley}, 0.2_{Lift})=0.6$ → the inference output of that context instance has an outcome equal to volleyball with membership value equal to 0.6.

Thus, a recommendation for users who would like to do sports is to play volleyball followed by swimming as an alternative according to the context values in this new instance.

In the implementation and evaluation chapter, I will discuss several applications that use this process of inference to decide actions and trigger events. These applications are more realistic and will emphasize the advantages of this method of inference to context-aware applications.

5.4 Summary

In Chapter 4, we presented the semantic modeling of context using ontologies to enhance the development of context-aware applications since it allows for the common understanding of context and facilitates reusability and interoperability. However, it also lacks the essential mechanisms to deal with uncertainty found in these types of context and falls short when reasoning about context information in the presence of uncertainty.

In this chapter, we presented a fuzzy reasoning approach that enables the infrastructure to leverage the semantically modeled contexts to include methods for representing, also using ontologies, the uncertainty in these types of context information.

The proposed fuzzy reasoning is an inductive reasoning approach that has the ability to discover the patterns in the context information used in the applications and automatically generates fuzzy rules that are used in the inference process.

We presented a detailed analysis of the inductive fuzzy reasoning mechanism. We described the process of fuzzifying context, elicitating membership functions and performing the fuzzy inference mechanism. We also illustrated this inference approach using a simple example to emphasize the novelty and effectiveness of this approach to deal with uncertainty in context-aware applications.

In the coming chapter, we will provide an in-depth discussion of the implementation of the different components of ACAI. We will also report on the experiments performed to evaluate the performance of the ACAI infrastructure both quantitatively as well as qualitatively.

Chapter 6

Implementation and Evaluation

6.1 Overview

This chapter describes the prototype implementation and evaluation of ACAI's core components for supporting context-awareness in pervasive environments. We will not discuss the implementation of the multiagent system as it was already discussed in Chapter 3.

ACAI prototype implements all the features described in Chapter 4 and Chapter 5. In specific, ACAI implementation had three phases. Figure 6-1 depicts these phases.

In the first phase, the ontology phase, context ontologies were developed and formulated using OWL [16]. Also in this phase, the repository system was developed. This repository provided a persistent storage facility for the developed ontologies and their instances that are used by applications built on top of ACAI.

The fuzzy inference ontology was also developed in this phase while its algorithms and implementation were extended to the second phase, the inference phase. The integration of fuzzy inference and context ontology was performed in the second phase as well.

The implementation of the context management agent and the inference agent, which we discussed in Chapter 3 were also part of this phase.

The prototype phase is the last phase. In this phase, I built the negotiation process, the remaining agents of the multiagent system, the APIs that are used to develop context-

aware applications and provide access to the repository. This was fallowed by the performance evaluation and testing.

ACAI's components are java-based. Agents were developed using JADE [51] and the core ontology functions were accessed using JENA [52]. In the coming sections, I will present, to an extent, the implementation details, obstacles we faced, and the methods we used to overcome these obstacles.

ACAI currently requires 649 KB of persistent storage. The size of the ontologies and instances varies according to the applications' configuration. They range from a few hundreds of bytes to tens of kilobytes for very detailed contextual information. The following sections describe in details the current version of ACAI and its implemented components.

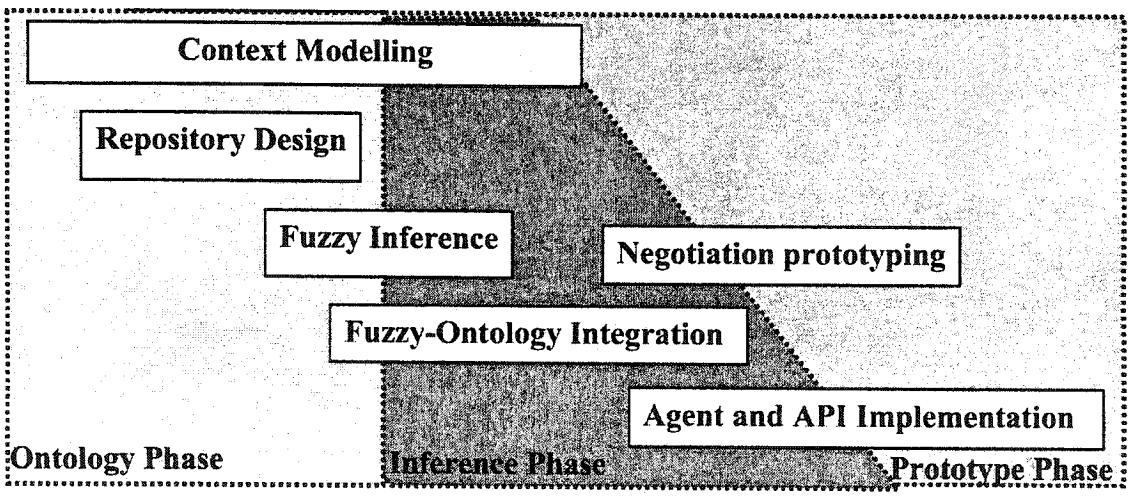


Figure 6-1 The three phases of the prototype implementation

6.2 Implementation of the Context Ontology

The proposed ontologies [48] were developed in two steps. The first step was to concrete the representation of the ontologies with a semantic language. We used OWL language for completing this step. The second step was to map these ontologies to java API so developers, using ACAI, can programmatically manipulate these ontologies. We therefore mapped the ontologies to java interfaces and implemented their corresponding concrete classes.

```
<owl:Class rdf:ID="PlaceInstance">
    <rdfs:subClassOf>
        <owl:Class rdf:about="#Structure"/>
    </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasAlternativeTrans">
    <rdfs:domain rdf:resource="#LocationDataType"/>
    <rdfs:range rdf:resource="#LocationDataType"/>
    <rdf:type rdf:resource="&OWL;TransitiveProperty"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasLocData">
    <rdfs:range rdf:resource="#LocationData"/>
    <rdfs:domain rdf:resource="#LocationDataType"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasLocationDataType">
    <owl:inverseOf>
        <owl:ObjectProperty rdf:about="#hasRelationToPlace"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#PlaceInstance"/>
    <rdfs:range rdf:resource="#LocationDataType"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasAccuracy">
    <rdfs:range>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&Fuzzy;FuzzyMembershipFn"/>
                <rdf:Description rdf:about="&Fuzzy;FuzzyNumber"/>
            </owl:unionOf>
        </owl:Class>
    </rdfs:range>
    <rdfs:domain rdf:resource="#LocationData"/>
</owl:ObjectProperty>
```

Figure 6-2 Snippet code of the location ontology

As an example, Figure 6-2 shows a snippet of the location ontology in OWL format. The snippet shows the class *PlaceInstance* and some of the properties associated with it. Developers can use any off-the-shelf editors to access the ontologies and define their instances, but this will be done off-line or during application design time. However, using our java implementation of the ontologies developers can easily manipulate and access ontologies at run time.

Figure 6-3 shows the UML [53] representation of some of the methods developed to manipulate this ontology. The figure also shows the associations this ontology has with

other java-implemented ontologies, such as the Similar class in the fuzzy ontology, to define the similarity between places, and the *GeneralDevice* class in the device ontology to define devices that might exist in this place instance.

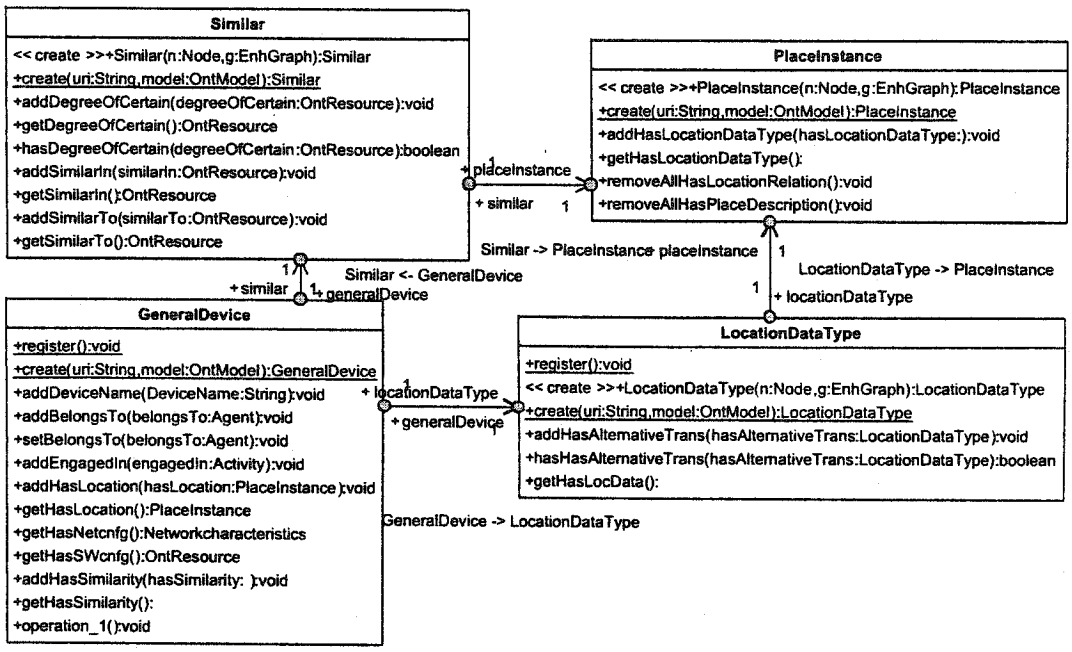


Figure 6-3 UML diagram of classes representing ontologies in ACAI

Likewise, all other ontologies were mapped to their corresponding java classes in order to provide runtime support for the manipulation of the semantically modeled context information. A validation process was performed on the developed ontologies before their concrete implementation. Ontologies were validated to ensure that they were well formed and consistence in structure and axioms with the W3 recommendations and requirements.

It is noteworthy that ACAI's ontologies had undergone several revisions and modifications in accordance with two guidelines. The first guideline addressed the concepts that should be defined in these ontologies. The second guideline addressed the methods of representing these concepts. These revisions took a considerable period of the thesis work, nearly one year of the thesis time, which proves that the hardest part in ontology design is to build a good ontology.

The first versions had vast changes in the concepts they provided while later ones were more stable and were only modified to provide finer details of these concepts. Also with

the introduction of the similarity and the alternative concepts, our ontologies became more stable in terms of the changes that occurred in their structure.

The current version of the ontologies has been stable for the past 10 months with no significant changes to most of the concepts defined in them.

The next step in the implementation was to provide a persistent repository for storing and querying ontologies and ontology instances.

6.3 Repository Implementation

The decision to use ontologies as the data model of ACAI, along with the high reliability and performance requirements, have imposed on us to develop a management system that supports temporal, as well as persistent, storage of information and provides an API to manipulate the information in both types of storage. Moreover, ACAI should provide an interface for developers and users to explore and browse the ontology repository. All these processes must be accomplished in a context-sensitive manner to reduce access overloading and increase system responsiveness.

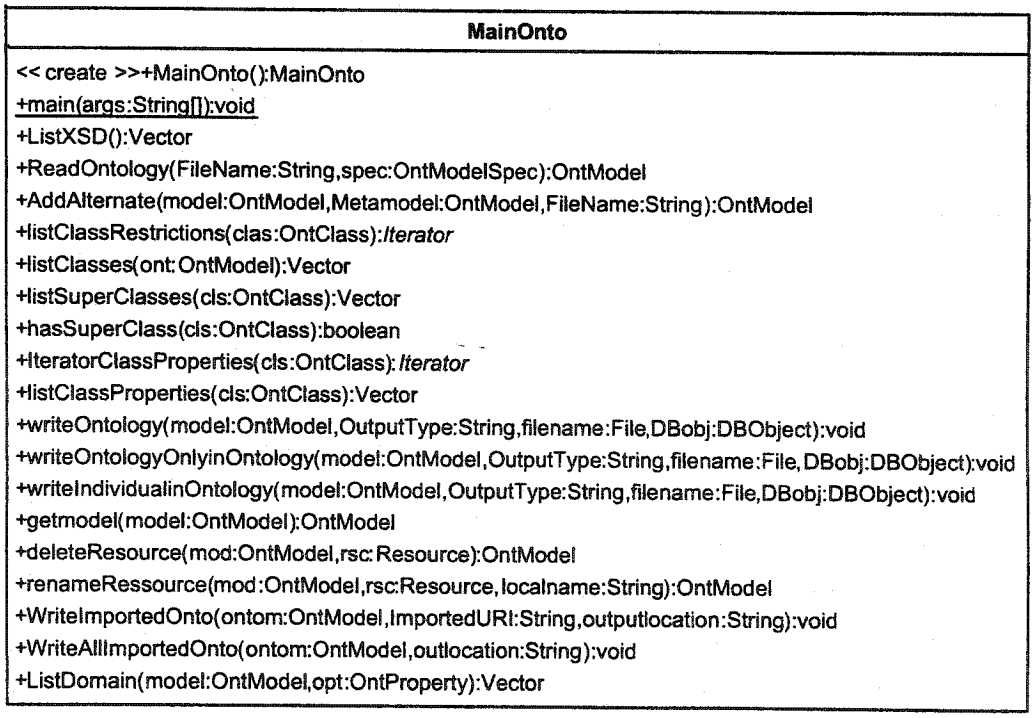


Figure 6-4 Excerpt of the methods used for ontology manipulation

Figure 6-4 shows a UML class diagram of the developed *Repository* class and some of the implemented methods that are used to manipulate the ontology information in both types of media. We used Jena [52] to develop this API. Some key methods in the API are the read/write methods and their variants. ACAI supports writing ontologies and their instances to the repository and/or to plain text files.

ACAI also provides the capability to separate ontologies from their instances. This, to the best of our knowledge, is not supported by available software for manipulating ontologies. This gives ACAI extreme flexibility in manipulating its ontologies since ontologies and their instances can be stored and read using different media simultaneously. In addition, the repository API supports the cashing of imported ontologies to the repository, which provides the runtime reliability and increases system responsiveness as required in design objectives of ACAI.

Other key methods in the repository API include the incremental addition, deletion and renaming of resources in the manipulated ontologies. These methods support the modification of resources on an individual basis during runtime.

The developed *Repository* API for manipulating ontology information has two goals. The first goal is to provide a small footprint API to access ontologies stored in the repository and the second goal is related to ACAI's inference engine, which I will describe in the coming section.

6.4 The Inference Engine Implementation

The implementation of the inference engine involves two processes. The first process addresses the fuzzification of the contextual information that is used in the inference mechanism. The second process addresses the concrete implementation of the algorithms and methods required to perform the inductive fuzzy reasoning.

As described in Chapter 5, the fuzzification process requires defining linguistic terms and elicitating membership functions. ACAI's API provides the required methods for performing these two steps.

Figure 6-5a shows an excerpt of the methods developed in the inference API for generating and handling fuzzy membership functions of the fuzzified ontologies. On the other hand, Figure 6-5b is an excerpt of the methods developed to manipulate these membership values. This includes methods for performing logical operations on these fuzzy membership functions such as the TNorm, TCoNorm and the Complement operations

To insure that the fuzzified context ontologies and inference outcomes are valid to their ranges and domains values, ACAI provides developers with methods for debugging and handling runtime exceptions. These methods report any invalidity or inconsistency in the inference outcomes if they breach their specifications in the corresponding ontology.

Figure 6-5c illustrates an excerpt of ValidityInference API and the methods developed to validate the inference outcome. Some key methods in this API are the ValidateDomain() and the ValidateRange() methods. The first method captures any violations in the inference outcome that does not comply with the domain specifications defined in the ontology under validation. The second method captures the invalidation that may occur when this outcome is not complying with the range specifications of the ontology under validation

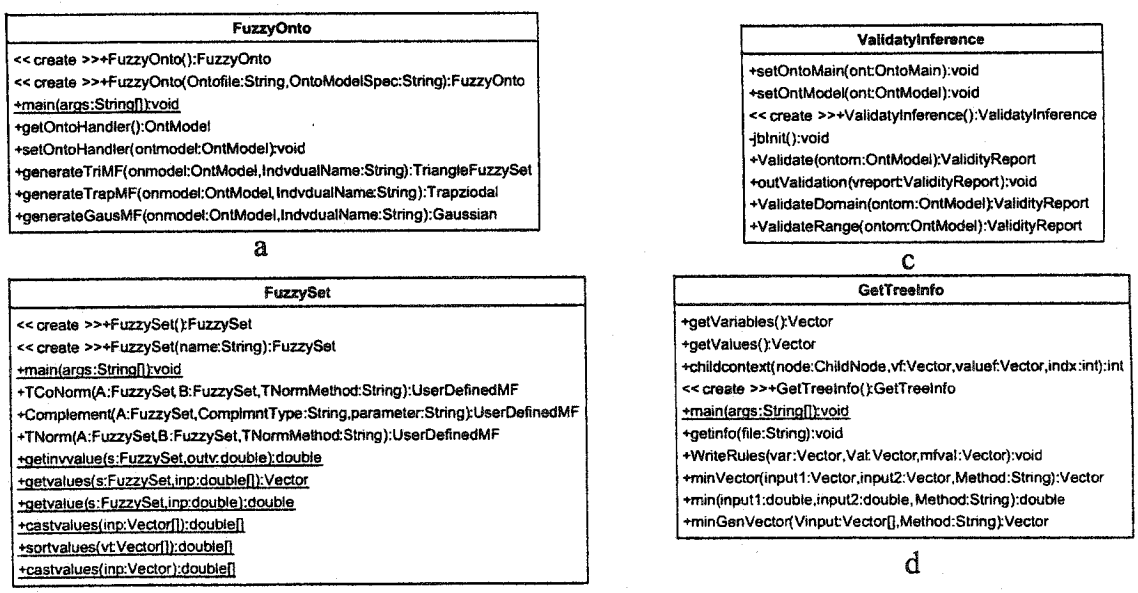


Figure 6-5 Excerpt of the inference engine implementation

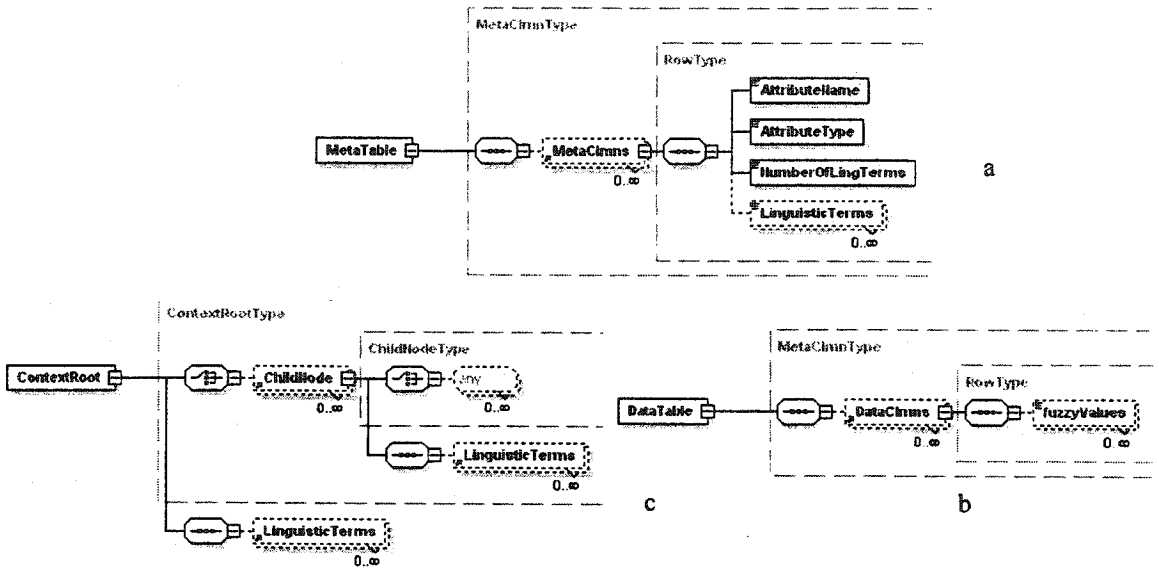
Finally using the methods shown in Figure 6-5d, developers can generate the rules that the inference agent will use in the inference process.

The internal inference process is hidden from developers, as they do not need to set or perform any custom changes to the reasoner internal behaviour. However, ACAI exposes some set/get functions for defining values needed in the inference process such as the cut-off value used in the pruning of the inference tree, the context information that is used in the inference process, and the reasoning method used in the inference process.

ACAI needs a repository facility to use in the inference process [114]. This is because the proposed reasoning mechanism is based on inductive fuzzy reasoning, fuzzy membership elicitation, tree generation, and rules extraction, which all require a working memory to store and manipulate the information in it. The current implementation of ACAI has three types of tables that ACAI uses in the inference process. These tables are the MetaTable, the DataTable and the TreeTable

Figure 6-6a depicts the structure of the MetaTable. The MetaTable is the table that holds information about the contexts used in the inference process. This includes contexts' names, types, and linguistic terms. The DataTable, shown in Figure 6-6b, is the table that holds the actual fuzzy values of the contexts defined in MetaTable table. Entries in the DataTable are the training examples used in building the inductive fuzzy tree and in the inference process.

Figure 6-6c depicts the structure of the TreeTable that represents the generated inductive fuzzy tree. The ContextRoot element is the root of the tree, and the ChildNode elements are the children of the root of tree. These ChildNode elements in turn are the parents of other ChildNode elements. Figure 6-6d is an excerpt of the fuzzy tree generated when we applied the inference process and the example discussed in section 5.3 of Chapter 5.




```
<ContextRoot Name="Temp" ParentName="none" LevelNumber="0" ContextName="Temp">
    <ChildNode Name="Look" ParentName="Temp" Betavalues="[0.274, 0.493, 0.233]"
 DatatableName="Data_Lv_1Nd_1">
        <ns1:ChildNode Name="Sport_Swim" ParentName="Look11" Betavalues="[0.351, 0.526, 0.123]"
DatatableName="Data_Lv_2Nd_1" xsi:type="ChildNode" />
        <ns2:ChildNode Name="Hum" ParentName="Look11" Betavalues="[0.316, 0.316, 0.368]"
DatatableName="Data_Lv_2Nd_2" xsi:type="ChildNode" >
            <ns2:ChildNode Name="Sport_Lift" ParentName="Hum21" Betavalues="[0.462, 0.000, 0.538]"
DatatableName="Data_Lv_3Nd_2" xsi:type="ChildNode"/>
            <ns2:ChildNode Name="Sport_Lift" ParentName="Hum21" Betavalues="[0.316, 0.316, 0.368]"
DatatableName="Data_Lv_3Nd_3" xsi:type="ChildNode"/>
            <LinguisticTerms Name="Hum_Hum"/>
            <LinguisticTerms Name="Hum_Norm"/>
        </ns2:ChildNode>
        <ns3:ChildNode Name="Sport_Lift" ParentName="Look11" Betavalues="[0.261, 0.000, 0.739]"
DatatableName="Data_Lv_2Nd_3" xsi:type="ChildNode" />
        <LinguisticTerms Name="Look_Sunny"/>
        <LinguisticTerms Name="Look_Cloud"/>
        <LinguisticTerms Name="Look_Rain"/>
    </ChildNode>
    <ChildNode Name="Wind" ParentName="Temp" Betavalues="[0.485, 0.106, 0.409]"
DatatableName="Data_Lv_1Nd_2">
        <ns4:ChildNode Name="Sport_Lift" ParentName="Wind12" Betavalues="[0.182, 0.000, 0.818]"
DatatableName="Data_Lv_2Nd_4" xsi:type="ChildNode" />
        <ns5:ChildNode Name="Sport_Volly" ParentName="Wind12" Betavalues="[0.696, 0.152, 0.152]"
DatatableName="Data_Lv_2Nd_5" xsi:type="ChildNode" />
        <LinguisticTerms Name="Wind_Wind"/>
        <LinguisticTerms Name="Wind_No"/>
    </ChildNode>
    <ChildNode Name="Sport_Lift" ParentName="Temp" Betavalues="[0.236, 0.000, 0.764]"
DatatableName="Data_Lv_1Nd_3"/>
    <LinguisticTerms Name="Temp_Hot"/>
    <LinguisticTerms Name="Temp_Mild"/>
    <LinguisticTerms Name="Temp_Cool"/>
</ContextRoot>
```

d

Figure 6-6 The inference engine implementation.
a) The MetaTable attributes structure. b) The DataTable attributes structure. c) The TreeTable attributes structure d) Excerpt of the generated fuzzy tree

6.5 Application Development Process

ACAI has the feature of relieving application developers from explicitly dealing with environment resources in order to accomplish their intended tasks and applications functionalities. This is because interactions with environment resources are delegated to ACAI's components discussed in Chapter 3.

Using ACAI, developers only need to follow a generic procedure for developing context-aware applications. This procedure is independent of ACAI internal structure and requires no prior knowledge of resources existing in the environment. The procedure is a sequence of steps that cover the three aspects supported by ACAI: modeling context, reasoning about context, and adapting to context.

In general, developing an application must have a decomposition step where developers state the functionalities provided by the application, the inputs needed and the expected outputs. This step is crucial to the success of any applications regardless of the framework or infrastructure used.

Using the results of this necessary step, developers will follow the following steps to develop applications using ACAI:

1- Select ontologies that best represent the input and output information of the application.

2- Decide classes and properties that will be used in each of these ontologies.

3- Create instances from these ontologies that correspond to the different entities and situations expected in the application.

4- State the inference process inputs such as membership functions elicitation, number of context to be used in the inference and the inference mechanism.

5- Define the inference tree input parameters and output conditions.

6- If required, state the negotiation input values, stopping criteria and utility functions used in the negotiation process.

Steps 1 to 3 address the semantic modeling of all types of information used in the application such as location, timing, users, services, etc. The third step could be implemented as a GUI interface provided to the users in order to set their contextual information such as activities and notification requirements.

Steps 4 and 5 instruct ACAI on the methods and mechanisms it can use for reasoning about context, trigging actions and delivering information.

Step 6 is needed if developers are using the adaptation mechanism of ACAI. They state the maximum and minimum values for contexts used in the negotiation, the negotiation process, and criteria of reaching an agreement.

In the coming sections I will present the implementation of the conferencing application developed using ACAI, the sub-applications that support the conferencing application and the experiments conducted to evaluate ACAI's performance.

6.6 The Conference Application

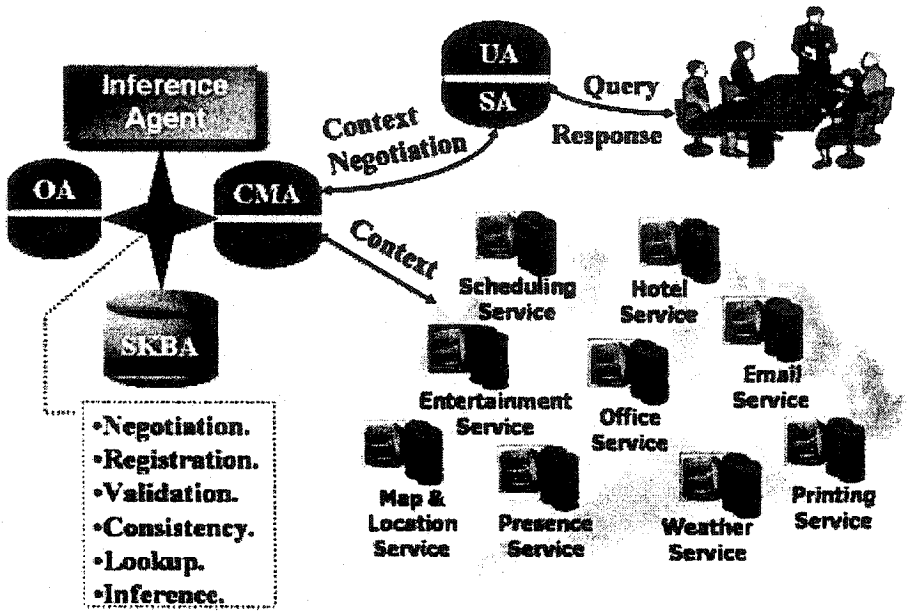


Figure 6-7 The conference environment and available services

In the previous sections, we described the detailed implementation of ACAI's components. Specifically, the semantic modeling of context discussed in Chapter 4 and the inductive fuzzy reasoning mechanism presented in Chapter 5.

In this section, I will provide insight on the applications that I and other researchers developed using ACAI and discuss the results of its thorough evaluation.

The aim of developing applications is to support the thesis that ACAI as an infrastructure provides developers with powerful abstractions that are easy to use for developing context-aware applications.

As for performance, I aim to validate the thesis that the novel approaches introduced in this research can be realized through lightweight implementation and provide good performance in terms of scalability, reliability, and effectiveness.

Using the developed ACAI agents, we implemented a context-aware conferencing environment capable of customizing conference services. This is based on captured context, attendees' profiles, and services' profiles defined according to the context

ontologies we have developed. The complexity of customizing the conference operation is greatly reduced by defining and executing contexts that are relevant to the attendees using the negotiation protocol discussed in Chapter 3.

Figure 6-7 shows the services and agents in the conference environment. We divided available services into two categories. These are *"services in advance"* such as hotel booking, scheduling, and entertainment and *"services on the fly"* such as the event-notification service, printing service and instant messaging service. The former category takes its input context from the attendees' profile provided during conference registration. The latter category takes its input context from the attendees' physical and logical context.

Agents are implemented using JADE [51], a publicly available agent platform. We extended the ACL package to include our negotiation performatives. The CMA provides APIs for registration, service selection, context specification, event notification, and event-based triggering methods. The OA provides APIs for browsing, navigating, searching, and validating ontology-based instances stored in a persistent repository and managed by the SKBA. The IA provides APIs for ontology-based inference such as subsumption, equivalence, restrictions, and consistency checking. It also provides the proposed inductive fuzzy inference mechanism to cope with vagueness and uncertainty in context information such as location proximity, presence range and service selection.

Our default deployment strategy is to install one CMA at each domain. A domain is either physical, like a building, or logical, like a network domain. We also deployed an inference agent at each domain, giving it the ability to clone itself on nearby hosts and form a federation in case of overloads.

6.6.1 Customizing the Conference Environment

Customizing the conference environment involves two phases: the registration phase and the service-provisioning phase. In the registration phase, attendees specify their context-based information in a user profile while they specify their "*services in advance*" requirements such as hotel location, price, and the length of their stay in a service profile. This information is sent to the CMA in a *CLS_Negotiation* message as shown in Figure

6-8. The CMA registers the user and sends his/her profile to the OA. The OA validates this profile against the ontology constructs and axioms defined in the context ontologies. It then sends the profile to the system knowledgebase agent to store it in the conference repository for further access and manipulation.

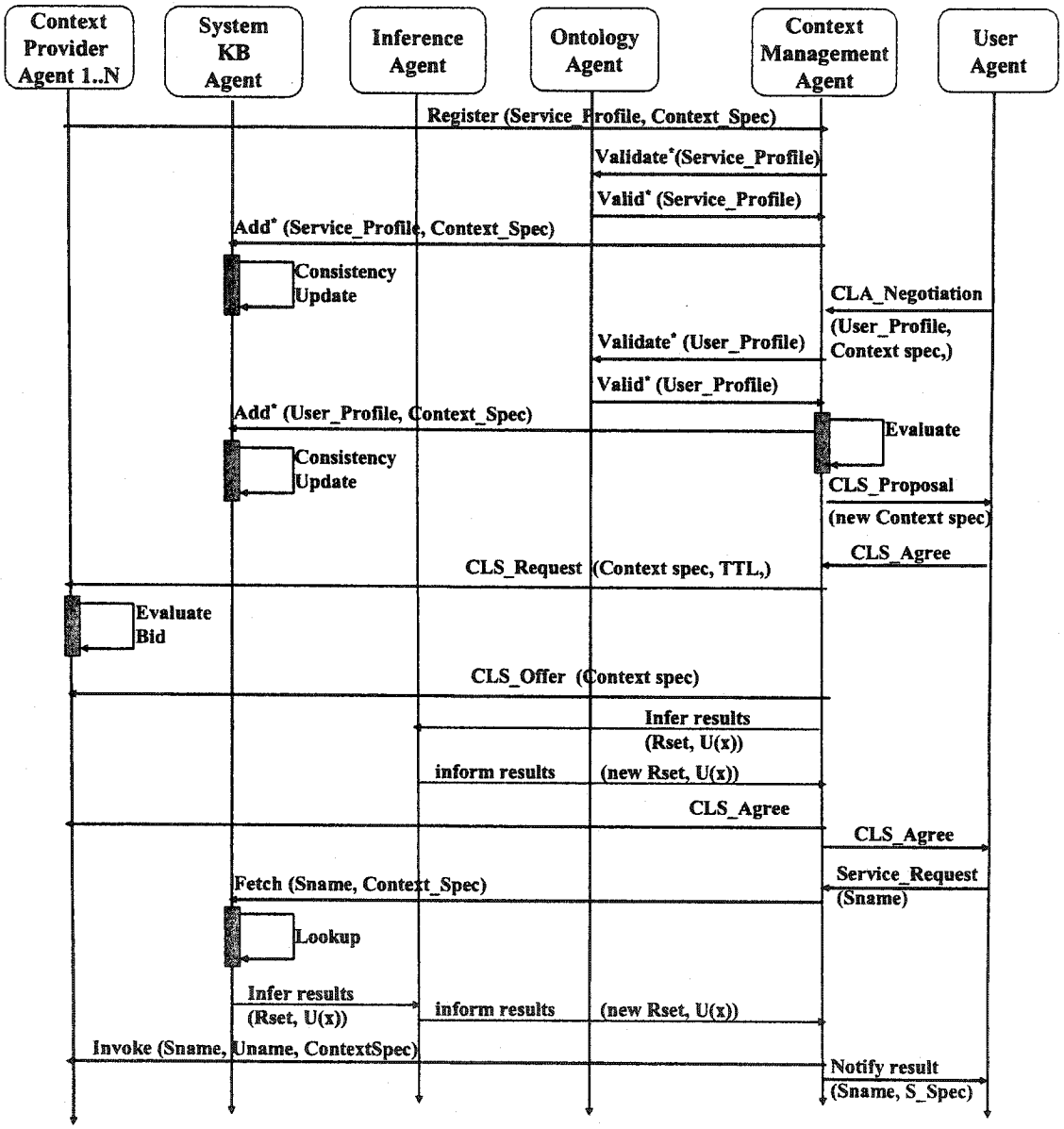


Figure 6-8 Messages sequence diagram for registration and negotiating context

After registration and validation, the CMA starts the second phase where it sends user's specifications to registered context providers to find services matching user requirements. The CPAs then reply with a *CLS_Offer* that includes in its content the available services

that are matching user specifications. The CMA sends these results to the inference agent to provide the closest match that maximizes user's utility function. Finally, after receiving the results from the inference agent, the CMA sends a *CLS_Agree* message to the user with these results in the message content.

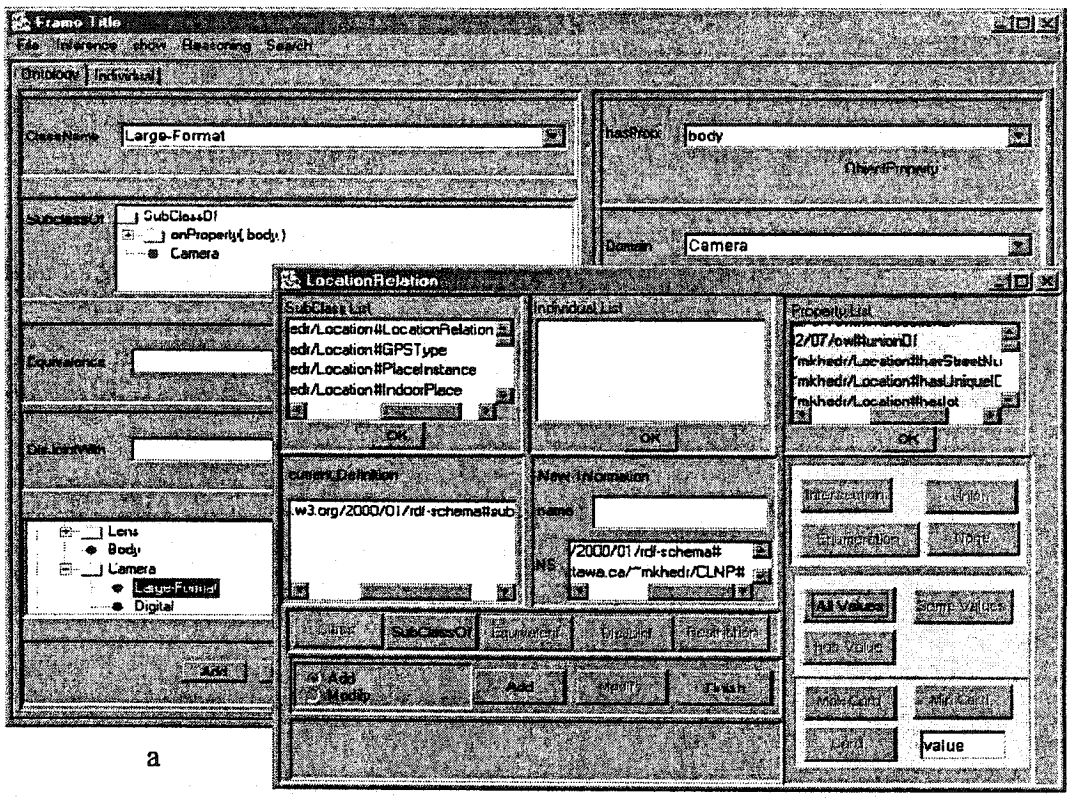
In the case of *"Services on the fly"*, the same message flow applies. However, *"Services on the fly"* have the ability to renegotiate context specifications and reason about received results at runtime. This is shown in Figure 6-8, starting from the service request message sent from the UA until the end of the message sequence diagram. The reason for this is that *"Services on the fly"* are dynamic, shared, and distributed all over the conference domain. Invoking them is highly dependent on current user context. Examples of these types of services are invoking the printer nearest to the user's location and with the least load or alerting attendees of some upcoming events.

We developed a set of GUI interfaces to simplify the process of modeling context using the developed ontologies and the process of inference using fuzzy logic. Developers can use these interfaces to write profiles compliant with our designed ontologies for services, devices, activities, and other contextual information in the conference application. At the same time, attendees can use these GUIs to write their individual profiles and activities

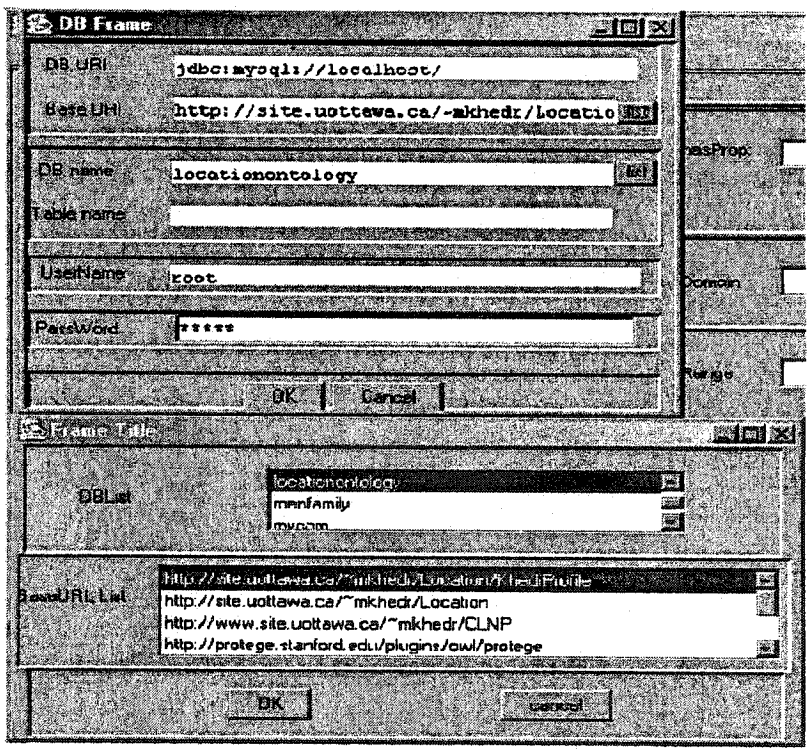
The GUI supports loading from and saving to files, databases, and remote URL locations. Figure 6-9a shows a snapshot of this GUI with the loaded ontology and methods of defining properties and instances using the loaded ontology. Figure 6-9b shows the interface used to load profiles and ontologies from the system knowledgebase repository.

The ontology agent maps the entries in the GUI to their corresponding ontology classes and properties and generates the OWL/RDF syntax for these defined profiles.

In the coming sections, I will present the printing service, the event-notification service, and the instant messaging service as examples of the different types of applications developed over ACAI infrastructure. I will present how these applications explored the aspects of semantic modelling, fuzzy inference, and adaptation through negotiation to provide their intended functionalities. Performance measurements will follow these illustrations as well as a critical overview about ACAI from the perspective of other researchers who used it in their projects.



a



b

Figure 6-9 GUI used for ontology editing and repository accessing

6.6.2 Context-Based Service Provisioning

The first type of applications that I will present deals with the process of discovering and using services in the conference environment. This service provisioning is based on users' requests, available services, and their associated contexts. The key concept that differentiates this process from other types of service discovery [8] is using ontologies for semantically modeling contextual information and fuzzy reasoning for inferring and matching services to users' requests.

The scenario used for this type of applications is identical to the one described in section 6.6.1. Users and services residing in an ACAI environment, such as our conference environment, issue a service request to the CMA. The CMA in return forwards the request to the SKBA for services lookup. The SKBA queries the repository for registered services that match the request. The returned results are represented in OWL format and are send to the IA for reasoning over them and providing the closest services matching users' requests. The IA applies one of the fuzzy reasoning mechanisms discussed in Chapter 5 and forwards the refined result to the CMA. The CMA replies to the user with the discovered service's location and preferences.

However, certain steps have to be done before this sequence of interactions takes place. They are the steps we discussed in section 6.5. First, the application developer has to decide the contexts to be used in the application. This is followed by stating and eliciting their linguistic terms and membership functions that the inference agent will use in the fuzzy inference process. The negotiation of the acceptable ranges of contexts follows so ACAI can adapt its service provisioning according to these ranges. Finally, at run time, contexts of entities are modelled using ontologies and reasoned about using the fuzzy inference.

6.6.2.1 Elicitation of the Fuzzy Membership Functions

In this step, developers specify the linguistic terms and their membership functions for the contexts used by the application. For example, we modeled the two types of contexts, (location and service load), used to lookup and invoke a printer in the conferencing

environment into three terms. The location has "near, within, far" while the service load has "low, medium, high" as their linguistic terms.

To elicit the membership functions of the context "location", we used the detection range of the RF-Tag system implemented in our MMARL lab at University of Ottawa as an indication of the location range. On the other hand, we used the number of documents in the printers' spool as indicator of the second context "service load". In numbers, the location range was taken from one to ten meters while the printer load was taken from zero to 20 documents.

As we mentioned in section 5.3, developers specify the membership values of some random samples and the IA elicits the membership functions that match these values using the Lagrange interpolation method. Figure 6-10 depicts this process for the linguistic terms "low" and medium" of the printer load. The dotted vertical lines are the samples provided by developers while the red lines with asterisks are the generated membership functions from these values. As shown from the figure, the first term "low" matches the trapezoidal membership function, the thick solid lines in the figure, while the term "medium" matches the Gaussian membership function.

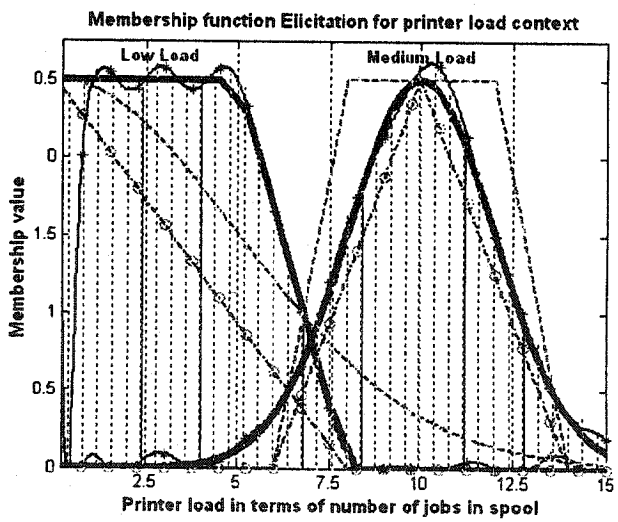


Figure 6-10 Elicitation of the service load membership functions

6.6.2.2 Negotiating context

After setting the context parameters and their membership functions, users and services can negotiate with ACAI the range of values of these contexts that suit them and their

situation. Providing a detailed illustration of the negotiation process is quite lengthy. Therefore, we will only show sample of the exchanged messages and the utility function used in the negotiation.

The following is an excerpt of the *CLS_RePropose* message that is sent from the user agent, *KhedrAgent*, to ACAI's CMA in response to a *CLS_Proposal* message. The excerpt shows the context parameter under negotiation, *LocationRange*, and the ranges required by the user. The excerpt also shows the rule, *Rule1*, that states the condition required to reach an agreement. The rule states that if the location range is between 6 and 8 meters then send a *CLS_Agree* message. Finally, the excerpt shows information about the utility function that is used to evaluate the concession and the satisfaction values of the context under negotiation.

```
<CLNP:CLNP rdf:ID="CLNP_khedr">
   <!--Context negotiation for location range-->
   <CLNP:hasRequester>
      <CLNP:UA rdf:ID="khedrAgent"/>
   </CLNP:hasRequester>
   <CLNP:hasContextSpec>
      <CLNP:ContextSpecification rdf:ID="Location" CLNP:hasContextElement="&Loc;PlaceInstance">
         <CLNP:hasContextParameter rdf:resource="#LocationRange"/>
      </CLNP:ContextSpecification>
   </CLNP:hasContextSpec>
   <CLNP:hasProcess>
      <CLNP:NegotiationMessage rdf:ID="CLS_RePropose" CLNP:TimeToLive="20ms">
         <CLNP:hasRule>
            <CLNP:NegotiationRules rdf:ID="Rule1">
               <CLNP:hasOperand rdf:resource="#and"/>
            </CLNP:NegotiationRules>
         </CLNP:hasRule>
      </CLNP:NegotiationMessage>
   </CLNP:hasProcess>
</CLNP:CLNP>


<CLNP:RuleOperand rdf:ID="and">
   <!--Rule used in Context negotiation-->
   <CLNP:hasAntecedent>
      <CLNP:Antecedent rdf:ID="bigger_than" CLNP:hasValue="6m">
         <CLNP:hasContextParameter>
            <CLNP:ContextParameter rdf:ID="LocationRange">
               <CLNP:usedByUtilityFn>
                  <CLNP:UtilityFunction rdf:ID="LogUF" CLNP:hasSenstivityparameter="5">
                     <CLNP:hasContextParameter rdf:resource="# LocationRange"/>
                  </CLNP:UtilityFunction>
               </CLNP:usedByUtilityFn>
            </CLNP:ContextParameter>
         </CLNP:hasContextParameter>
      </CLNP:Antecedent>
   </CLNP:hasAntecedent>
   <CLNP:hasAntecedent>
      <CLNP:Antecedent rdf:ID="less_than" CLNP:hasValue="8m">
         <CLNP:hasContextParameter rdf:resource="# LocationRange "/>
      </CLNP:Antecedent>
```

```
        </CLNP:hasAntecedent>
        <CLNP:hasConsequent>
            <CLNP:Consequent rdf:ID="CLS_Agree">
                <CLNP:hasContextParameter rdf:resource="# LocationRange "/>
            </CLNP:Consequent>
        </CLNP:hasConsequent>
    </CLNP:RuleOperand>
```

Figure 6-11a depicts the utility function used in this application. We used the logarithmic function discussed in section 3.5.2.3, where we set the minimum and maximum values to correspond to the ranges of the location and service load contexts.

The zigzagged black region in Figure 6-11b is the potential region where an agreement can be reached between what users' require and what context providers can support. The dotted line is the actual agreement reached between the agents for the abovementioned negotiation instance. This instance of agreement has location range between 6 and 8 meters and service load between 5 and 7 documents.

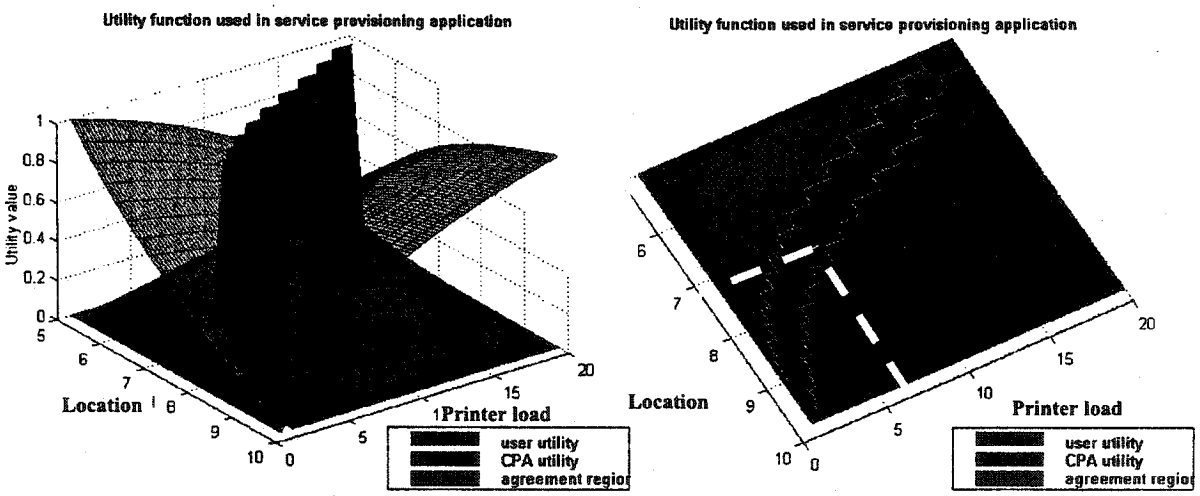


Figure 6-11 Utility function used in context negotiation.
a) 3-D plot of the utility function. b) Region of agreement.

6.6.2.3 Context Modeling and Inference

Figure 6-12a shows the semantic representation of context information and the inference made by the IA about users' location and received services from the SKBA. The figure shows that user "khedr" has *actorLocation* property with value equal to "A504". This information is taken as a fact in the reasoning process.

The IA uses the Location ontology to deduce that

1. *Khedr* has *actorLocation A504* → Fact 1
2. *A504* is a room → direct instance 1
3. *A504* is an inside place → Subsumption deduction 1.
4. *University of Ottawa* is a University → direct instance 2.
5. *University of Ottawa* has inside place *A504* → Fact 2.
6. *Khedr has actorLocation University of Ottawa* → deduced from Fact 1 and Fact 2.
7. *MMARL sameAs A504* → Equivalence deduction 1.
8. *Khedr has actorLocation MMARL* → deduced from Equivalence deduction 1 and the Fact 1.

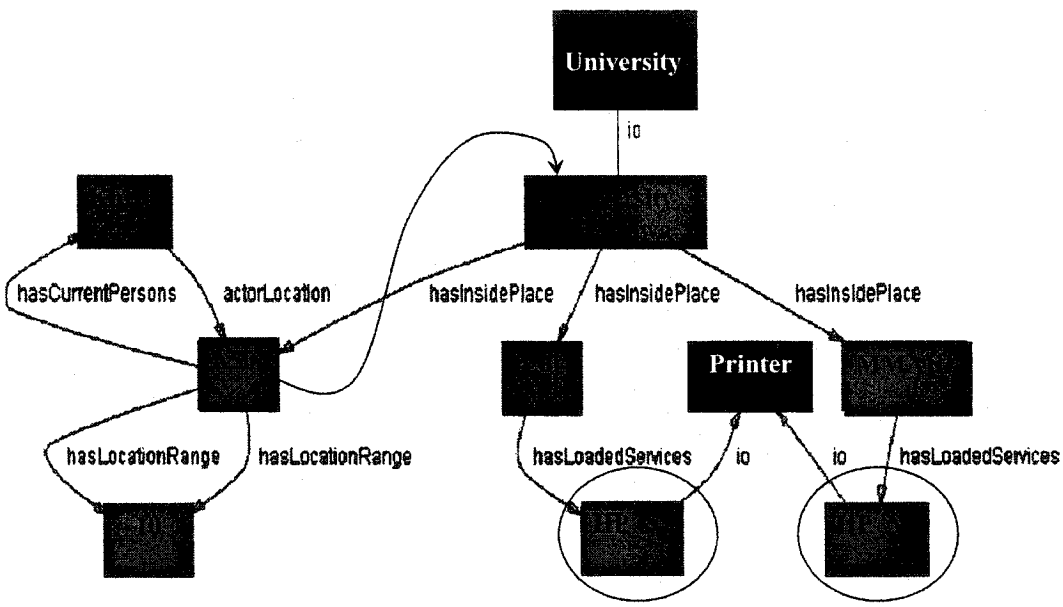


Figure 6-12 a The semantic representation of a user's context.

With this chain of reasoning, the IA can successfully select the available services in A504, which is equivalent to MMARL, and match user requests.

Figure 6-12b shows the fuzzy inference mechanism used by the IA to filter the set of available services into a smaller set that resembles the user specifications of the printer. Notice that there are printers HP_N2 and HP_N3 that are used in the fuzzy reasoning. These printers came from the facts:

- A504 has the property *hasLocationRange* that is equal to 10 meters.
- B502 and B504 where these printers reside are within this range

- B502 and B504 are within the range of 6-8 meters that was negotiated for.

More surprisingly is the returned service from the inference process, printer HP_N3, which is not located in the same place where the user is. The user is in A504 while HP_N3 is in B502. This can be traced back using the reasoning debugging option supported by ACAI. The IA used the compositional rule of inference as the method of inference. In other words, it provided the printer that best matches the aggregated specifications provided in the query.

It was found that HP_N3 was actually the best printer to project to the user as it was within the negotiated context, 6-8m, and have a high membership value in the "*Low printer load*" membership function.

ID	Location	MFL	Load	MFLD	Fuzzy Inference
HP_N2	B504	0.1	Low	0.9	0.1
HP_N1	MMARL	0.8	Low	0.3	0.3
HP_N3	B502	0.5	Low	0.7	0.5

Figure 6-12 b The fuzzy inference process

6.6.3 Event-Notification Service

```
<Person rdf:ID="Roger">
  <hasContactInfo rdf:resource="#RGContactInfo"/>
  <name>Roger</name>
  <family_name rdf:resource="#McGrath"/>
  <hasActivity rdf:resource="#RGActivity"/>
  <Loc:hasPlaceInstance rdf:resource="&Loc;B502"/>
  <hasAgent rdf:resource="#RGAgent"/>
 </Person>

 <AgentContactInfo rdf:ID="RGContactInfo">
  <mbox>RG@uottawa.ca</mbox>
 </AgentContactInfo>

 <Literal rdf:ID="McGrath"/>

 <Contivity:Activity rdf:ID="RGActivity">
  <ConTime:hasTimeParameter rdf:resource="&ConTime;meetingtime"/>
  <Loc:hasPlaceInstance rdf:resource="&Loc,MMARL"/>
 </Contivity:Activity>

<SoftwareAgent rdf:ID="RGAgent">
  <preferences rdf:resource="&SoftwareProfile;JADE"/>
  <name>RG@SKDanah:1099</name>
 </SoftwareAgent>

<StandardTime rdf:ID="meetingtime">
  <hasTimevalue>2004-01-27T12:05:00 </hasTimevalue>
</StandardTime>
```

Excerpt of the user profile sent from the UA to the CMA for subscribing to an event-notification service

Figure 6-13 Excerpt of the user profile

The second type of applications that we implemented using ACAI deals with event-monitoring and notification.

The snippet shown in Figure 6-13 is an excerpt of a user profile in OWL format, that includes his contact information modeled using the ConFoaf ontology, his activity information modeled using the Contivity ontology, the schedule of these activities modeled using the time ontology, and the location of the user modelled using the location ontology. The user agent sends the user profile to the CMA in an Agent communication Language (ACL) message. The ACL message is of type FIPA *Subscribe* performative in order to inform ACAI to add a monitoring event service for this user.

The CMA extracts the scheduling information from the user profile and sends it to both, the SKBA to store it in the system repository and to the IA to reason about user's context and notify the user about his scheduled events. The reasoning process is based on fuzzy logic because the two contexts required in the reasoning process are vague in nature, location, and time. Moreover, the process of notification is not crisp but has a range of transition in its outcome.

The same steps that we discussed in the previous section for developing context-aware applications are also used in this application. However, we only present the inference process since the negotiation mechanism is lengthy and nearly identical to what we presented in section 6.6.2.2.

6.6.3.1 Fuzzy Inference in the Event-Notification Service

From study, location as a context can be represented as a position and a direction. Position was mapped to {near, within, far} linguistic terms while direction was mapped to {left and right}. This produced five linguistic terms that are shown in Figure 6-14. In addition, users specify time in standard format, example notify me at 12:00PM. However, if he is far from the location where this event will take place, he might be delighted to be informed before 12:00 PM so he can catch the event in time. Thus we fuzzified the time values to five linguistic terms, {too early, early, about to start, late, very late}.

Finally, the action taken by the notification service is itself uncertain. The user may specify to show text on his PDA as a kind of alert but he might be already late for his

scheduled event and thus playing a tone would be more applicable to get his attention. Thus, we modeled the alert process to the following linguistic terms {no notification, low notification, medium notification, and high notification}. The IA uses the compositional rule of inference method to combine these inputs that represent the contextual information needed in this service and generates the rule surface diagram shown in Figure 6-14. It uses this rule surface to decide the best action to take for alerting the subscribed users about their events according to their current locations and the time of these events

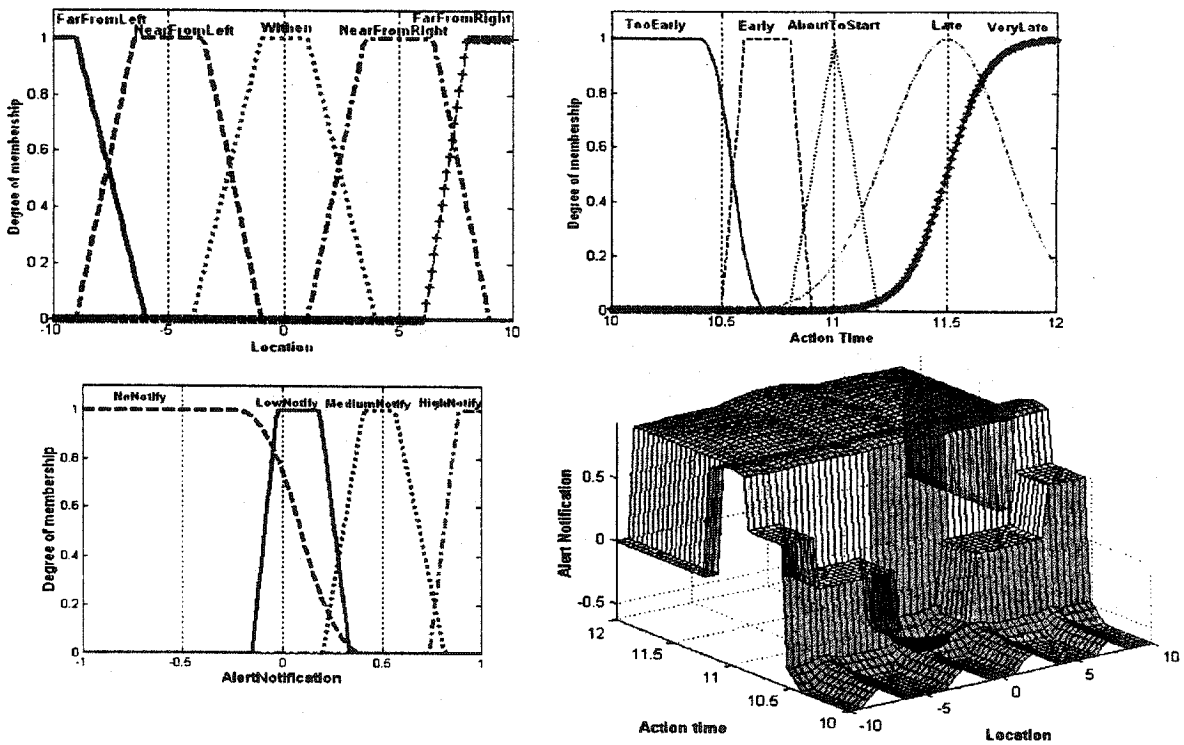


Figure 6-14 Membership functions and rule surface of the event-notification service

6.6.3.2 Semantic Inference in the Event-Notification Service

For the fuzzy inference to take place, the IA has first to reason semantically over these contexts to deduce the appropriate contexts that will be used in the fuzzy inference.

To illustrate this, a user in the deployed conferencing environment may issue an event-notification request that requires ACAI to notify him about the *MMARL_Meeting* event that will take place in the *MMARL_LAB*.

ACAI will use the abovementioned fuzzy inference if it has the fact that the user is in *MMARL_LAB* and that the *MMARL_Meeting* will take place in that location. The problem is that ACAI's knowledge about this event and its location differs from what the user issued in his request.

The following is a snapshot from the knowledgebase repository about the event, user location, and event location.

```
    <!--CMA knows that this is the location where MMARL_Meeting is taking place-->
<Loc:Complex rdf:about="&Loc;B502">
    <Loc:hasLocationRole>
        <Loc:Educational rdf:about="&Loc;MMARL_Meeting">
            <rdf:type rdf:resource="&Loc;Complex"/>
        </Loc:Educational>
    </Loc:hasLocationRole>
    <Loc:hasLocationDataType>
        <Loc:CartesianType rdf:about="&Loc;SiteCartesian">
            <Loc:hasLocData>
                <Loc:LocationData rdf:about="&Loc;SiteLocData">
                    <Loc:hasHorizontal>1250</Loc:hasHorizontal>
                    <Loc:hasVertical>2000</Loc:hasVertical>
                </Loc:LocationData>
            </Loc:hasLocData>
        </Loc:CartesianType>
    </Loc:hasLocationDataType>
</Loc:Complex>

    <!--the user knows that this is the location where MMARL_Meeting is taking place-->
<Loc:Indoor rdf:about="&Loc;MMARL_LAB">
    <Loc:hasLocationDataType rdf:resource="&Loc;SiteCartesian"/>
</Loc:Indoor>
    <!--CMA knows that this is the location where the user is currently in-->
<Loc:Office rdf:about="&Loc;Office@5thFloor">
    <Loc:hasLocationDataType>
        <Loc:VectorType rdf:about="&Loc;CLbyVectorType">
            <Loc:hasAlternativeTrans rdf:resource="&Loc;SiteCartesian"/>
        </Loc:VectorType>
    </Loc:hasLocationDataType>
</Loc:Office>
```

As shown from the repository snapshot, the *MMARL_Meeting* will take place in *B502* and not in *MMARL_LAB*. It also shows the location of the user is *Office@5thFloor* and not *MMARL_LAB*. Thus, ACAI is required to infer and relate these locations and events semantically before it can apply the fuzzy inference process. As a result, the IA has to prove that:

- *MMARL_LAB* is the same as *B502*
- *MMARL_Meeting* will take place in *MMARL_LAB*,
- User location is the same as *B502*.

This is shown in the following reasoning snapshot

***begin derivation of [Loc:MMARL_LAB, owl:sameAs, Loc:B502]

Rule **RULE_LOC_SameAS_LocData**

concluded (Loc:MMARL_LAB owl:sameAs Loc:B502) <-

Fact (Loc:MMARL_LAB rdf:type Loc:PlaceInstance)

Fact (Loc:B502 rdf:type Loc:PlaceInstance)

Fact (Loc:MMARL_LAB Loc:hasLocationDataType Loc:SiteCartesian)

Fact (Loc:B502 Loc:hasLocationDataType Loc:SiteCartesian)

Fact (Loc:SiteCartesian Loc:hasLocData Loc:SiteLocData)

Fact (Loc:SiteCartesian Loc:hasLocData Loc:SiteLocData)

***end derivation of [Loc:MMARL_LAB, owl:sameAs, Loc:B502]

***begin derivation of [Loc:MMARL_LAB, Loc:hasLocationRole, Loc:MMARL_Meeting]

Rule **RULE_LOC_SameAS_LocRole**

concluded (Loc:MMARL_LAB Loc:hasLocationRole Loc:MMARL_Meeting) <-

Fact (Loc:B502 rdf:type Loc:PlaceInstance)

Fact (Loc:MMARL_LAB rdf:type Loc:PlaceInstance)

Rule **RULE_LOC_SameAS_DataType**

concluded (Loc:B502 owl:sameAs Loc:MMARL_LAB) <-

Fact (Loc:B502 rdf:type Loc:PlaceInstance)

Fact (Loc:MMARL_LAB rdf:type Loc:PlaceInstance)

Fact (Loc:B502 Loc:hasLocationDataType Loc:SiteCartesian)

Fact (Loc:MMARL_LAB Loc:hasLocationDataType Loc:SiteCartesian)

Fact (Loc:B502 Loc:hasLocationRole Loc:MMARL_Meeting)

***end derivation of [Loc:MMARL_LAB, Loc:hasLocationRole, Loc:MMARL_Meeting]

***begin derivation of [Loc:Office@5thFloor, owl:sameAs, Loc:MMARL_LAB]

Rule **RULE_LOC_SameAs_Alternative**

concluded (Loc:Office@5thFloor owl:sameAs Loc:MMARL_LAB) <-

Fact (Loc:MMARL_LAB Loc:hasLocationDataType Loc:SiteCartesian)

Fact (Loc:Office@5thFloor Loc:hasLocationDataType Loc:CLbyVectorType)

Fact (Loc:CLbyVectorType Loc:hasAlternativeTrans Loc:SiteCartesian)

****end derivation of [Loc:Office@5thFloor, owl:sameAs, Loc:MMARL_LAB]

From these chains of semantic reasoning about location and event contexts, the inference agent can successfully proceed and reason about the uncertainty in these contexts using the fuzzy inference.

6.6.4 The Instant Messaging Application

The last type of applications that we implemented using ACAI is an instant messaging system (IMS). IMS is an example of the applications that semantically negotiate modeled context and use the inductive fuzzy inductive reasoning as its inference mechanism.

IMS is an extended version of the buddyspace open source [56] where we wrapped its interaction mechanism with agent technology and we modified its internal structure to provide contextual information about users and services registered with our ACAI.

IMS provides tailored contextual information such as location-awareness, presence-awareness, event monitoring, activity notification and more based on the fuzzified semantic context information and rules generated form the inductive fuzzy tree.

Figure 6-15 portrays our proposed instant messaging system, context sources, context interpretation and inference process. The proposed IMS is composed of autonomous agents that act on user's behalf, represent services, and wrap system's functionalities. The users lunch their user agents explicitly, while the context management agent activates the services and system agents. Every agent in the IMS has control over its behaviour and can make decisions autonomously. Agent behaviours are enriched with semantic rules to increase thier pro-activeness when facing new situations that might rise due to users' mobility, devices' capabilities, and frequent changes in the environment.

The IMS uses an RF-code[3] system for acquiring contextual information. We extended the RF-tags to provide other contexts besides identity and location. Taking advantage of the unique ID value of each tag, ontology profiles of entities are indexed in the system knowledgebase repository using these ID values. These profiles include implicit contextual information such as activities of users, preferences of services and rules generated for each entity holding a tag in the environment.

Figure 6-15 shows the categories we developed for the different entities in the environment and how they are associated to the tag IDs. For example, when the CMA detects a tag with ID 11175, it automatically interprets this tag to be a device of type

---

[3] www.rfcode.com

laptop. In addition, by detecting the tag, the CMA knows the current location of the laptop and can fetch its OWL profile from the repository to gain knowledge about its implicit context such as software and hardware capabilities, owner, and rules governing its execution

The IA uses these OWL profiles along with the inference process, shown at the top left of Figure 6-15 and explained in Chapter 5, to build the inductive fuzzy tree. After building the tree, the IA uses this tree to extract features and patterns embedded in the contextual information, and to generate the rules used in monitoring and governing the behaviour of applications and the interactions among entities in the IMS system.

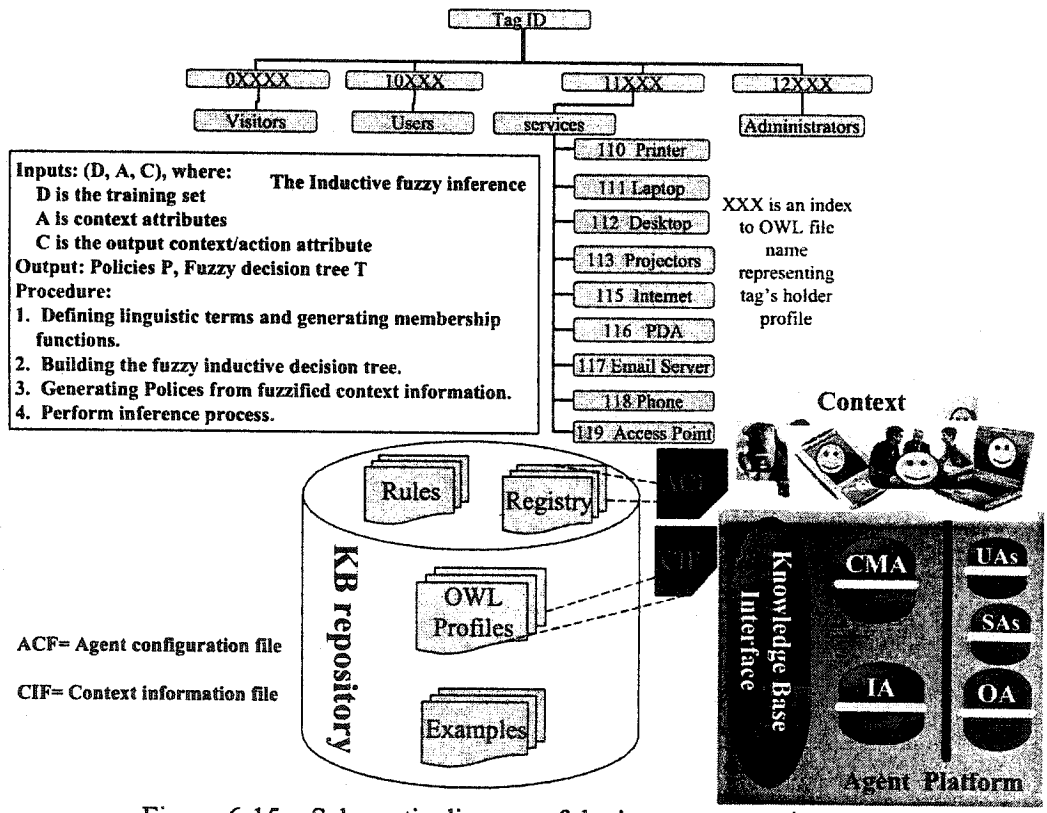


Figure 6-15 Schematic diagram of the instant messaging system

Every entity in the environment, user or service, has two configuration files: the Agent configuration File (ACF) and the Context Information File (CIF).

ACF holds information about the agent that works on behalf of the entity. This information includes the user contact information and reference to the location of his CIF

profile or the service description and reference to the CIF profile if the entity is a service. Figure 6-16 is an excerpt of the ACF and CIF of a printing service in the environment while Figure 6-17 is an excerpt of a user's ACF file. The CMA uses these files to customize the IMS client software, to register agents with the system, to lookup services and to enforce and manage interaction between entities that are using the instant messaging.

```
<AgentConfiguration
BSConfFile=".\BSprinter.conf"
ServConfFile=".\printer.xml"
TableName="agentinfo303">
 <Services Type="Printer"
   name="MMARL Printer">
  <Owner>MMARL</Owner>
  <Ontology>ConOnto</Ontology>
  <Protocol>CLNP</Protocol>
  <AgentRole>Service</AgentRole>
  <Registered>true</Registered>
  <MaxClients>1</MaxClients>
  <OWLConfiguarationFile>.\\printer.owl
  </OWLConfiguarationFile>
 </Services>
</AgentConfiguration>
```

```
<Context:ServiceProfile rdf:ID="HPLaserJetProfile">
 <Context:hasNetwrokSupport rdf:resource="#100M_Ethernet"/>
 <Context:hasServiceLocation rdf:resource="#B502"/>
 <Context:hasServiceParameter rdf:resource="#Color"/>
 <Context:hasServiceParameter rdf:resource="#PaperSize"/>
 <Context:hasserviceActor rdf:resource="#PrinterAgent"/>
 <Context:hasServiceParameter rdf:resource="#Resolution"/>
</Context:ServiceProfile>
<Context:Agent rdf:ID="PrinterAgent">
</Context:Agent>
```

Excerpt of the ACF and CIF of the printing service.

Figure 6-16 The ACF and CIF file

```
<AgentConfiguration
BSConfFile=".\BShamid.conf"
UserConfFile=".\hamid.xml"
TableName="agentinfo303">
 <Services Type="User" name="HamidAgent">
  <Owner>H.Harroud</Owner>
  <Ontology>ConOnto</Ontology>
  <Protocol>CLNP</Protocol>
  <AgentRole>User</AgentRole>
  <Registered>true</Registered>
  <OWLConfiguarationFile>.\\harroud.owl
 </OWLConfiguarationFile>
 </Services>
 <Services Type="Service Agent" name="PdfPrinteragent">
     <Owner>H.Harroud</Owner>
     <Ontology>ConOnto</Ontology>
     <Protocol>CLNP</Protocol>
     <AgentRole>PDF Printer service</AgentRole>
     <Registered>true</Registered>
     <MaxClients>1</MaxClients>
     <OWLConfiguarationFile>.\\pdfPrinter.wol
    </OWLConfiguarationFile>
 </Services>
</AgentConfiguration>
```

```
<UserConfiguration>
  <ImageIcons>
   <OnLine>images/hamid.gif</OnLine>
   <OffLine>images/hamid_red.gif</OffLine>
    <OnPhone>images/dock.gif</OnPhone>
     <Away>images/hamid_U.gif</Away>
     <Meeting>images/conf.gif</Meeting>
     <Launch>images/rest.gif</Launch>
     <Bussy>images/offline_msg.gif</Bussy>
     <Chat>images/chat.gif</Chat>
   </ImageIcons>
</UserConfiguration>
```

The figure to the left shows the parameters defined in Hamid ACF file while the above excerpt is the personalized icons used by Hamid in his IMS client.

Figure 6-17 The ACF file of a user in the instant messaging application

6.6.4.1 Negotiating Context in the IMS System

The sequence of messages used to negotiate context in the IMS system is identical to the sequence diagram shown in Figure 3-10 and explained in section 3.5.2.2.

Table 6-1 is an excerpt of the *CLS_Negotiation* and *CLS_RePropose* messages used in the IMS application. The *CLS_Negotiation* message shows the requester, *KhedrAgent*, the context manager agent, *MMARL_CMA*, and the context specification, *Presence*, that is used in this negotiation instance.

KhedrAgent is negotiating the minimum and maximum distances that IMS should use to project his presence to other entities in his buddy-list. This is to occur during the hours of 9:00 am to 12:00 pm, while he is in his 5th floor office in the Colonel-By building at the University of Ottawa campus. The *CLS_RePropose* message shows that it is a response to the *CLS_Proposal* (not shown for space limitations), and it includes an agreement rule, *AgreementRule*, that states:

"If *User_Presence* will not be projected to *MIRLGroup* then send a *CLS_Agree*"

The rule indicates that *KhedrAgent* will agree on the new values of the context specification being negotiated if the *MMARL_CMA* guarantees not to project his presence to users registered in the *MIRLGroup*.

Of course, this requires the *MMARL_CMA* to monitor *KhedrAgent's* location, the time, and individuals that are registered with the *MIRLGroup*. It must also cope with any inaccuracy in the context. This is achieved using the IFR approach.

It should be mentioned that IFR is the general case of inductive reasoning. IFR becomes a standard inductive reasoning by making the values of all fuzzy membership functions equal to one. Thus, our approach is generally applicable to deterministic situations as well. However, this requires the assurance that captured contexts are highly accurate, and contain no ambiguity in their meanings.

CLS_Negotiate	CLS_RePropose
<!-- Starting of a CLNP negotiation --> <CLNP rdf:ID="Khedr_CLNP_Negotiation"> <hasManager rdf:resource="#MMARL_CMA"/> <hasRequester rdf:resource="#khedrAgent"/> <hasProcess rdf:resource="#CLS_Negotiate"/> </CLNP> <!-- Type of negotiation message --> <NegotiationMessage rdf:ID="CLS_Negotiate"> <hasContextSpec> <!-- negotiationing presence --> <ContextSpecification rdf:ID="Presence"> <hasContextParameter> <!-- minimum value --> <Context zParameter rdf:ID="min"> <hasValue>5 meters</hasValue> <hasType rdf:resource="#Floor_5"/> <hasUtilityValue>1.0</hasUtilityValue> </ContextParameter> </hasContextParameter> <!-- rest of information .. not shown --> <hasContextElement rdf:resource="#User_Presence"/> </ContextSpecification> </hasContextSpec> </NegotiationMessage> <!-- when the negotiation is effective --> <Contivity:Activity rdf:ID="User_Presence"> <ConTime:hasTemporalDescription> <ConTime:TemporalDescription rdf:ID="Morning"> <Fuzz:hasEndPoint>12.00</Fuzz:hasEndPoint> <Fuzz:hasAssociatedFuzzyTerm rdf:resource="#morningMF"/> <ConTime:hasUnit>hours</ConTime:hasUnit> <Fuzz:hasStartPoint>9.00</Fuzz:hasStartPoint> </ConTime:TemporalDescription> </ConTime:hasTemporalDescription> <Loc:belongsto> <Loc:Floor rdf:ID="Floor_5"> <Loc:belongsto> <Loc:Building rdf:ID="ColonlBy"/> </Loc:belongsto> </Loc:Floor> </Loc:belongsto> </Contivity:Activity>	<!-- Starting of a CLNP Re-Propose--> <CLNP rdf:ID="Khedr_response"> <hasProcess rdf:resource="#CLS_RePropose"/> </CLNP> <NegotiationMessage rdf:ID="CLS_RePropose"> <!-- Indicating which message this instance is replying to--> <inResponseTo rdf:resource="#CLS_Proposal"/> <hasContextSpec rdf:resource="#Presence"/> <hasRule rdf:resource="#AgreementRule"/> <messageType>CLS_RePropose</messageType> </NegotiationMessage> <!-- The Rule that defines the agreement condition--> <NegotiationRules rdf:ID="AgreementRule"> <hasOperand rdf:resource="#And"/> </NegotiationRules> <RuleOperand rdf:ID="And"> <ConRule:hasAntecedent> <ConRule:Antecedent rdf:ID="agreementAntc"> <ConRule:entityB> <Confoaf:Group rdf:ID="MIRLGroup"/> </ConRule:entityB> <ConRule:relation rdf:resource="&ConPolicy;Negative"/> <ConRule:EntityA rdf:resource="#User_Presence"/> </ConRule:Antecedent> </ConRule:hasAntecedent> <ConRule:ofPriority rdf:resource="&ConRule;Required"/> <ConRule:hasConsequence> <ConRule:Consequent rdf:ID="agreementCons"> <ConRule:entityB> <NegotiationMessage rdf:ID="CLS_Agree"/> </ConRule:entityB> </ConRule:Consequent> </ConRule:hasConsequence> </RuleOperand>

Table 6-1 Excerpt of the negotiation messages used in the IMS system

6.6.4.2 The Inductive Fuzzy Inference in the IMS System

The application developer provides the fuzzification of the semantic contexts used in the IMS service. These fuzzified contexts are stored in the knowledgebase repository and used to generate the initial training examples needed in building the inductive fuzzy tree. Table 6-2 illustrates these steps.

The first row represents the three types of context used in IMS to deduce the presence of entities in the environment. The second row shows the membership functions used to fuzzify these three contexts. As application developers, we divided the location information of the fifth floor in Colonel-By building at university of Ottawa with respect to our lab into three categories (near, within and far). The IA then chose the respective membership functions of these three categories according to the algorithm described in Chapter 5. The same process applies to the other two contexts as shown in the second row of Table 6-2

Rows 3 to 8 illustrate samples of the training examples used by the inference agent to build the inductive fuzzy tree (IFT). These values are acquired from previous records of students' activities in our lab. The inference agent builds the IFT, (Row 9, Column 1) in Table 6-2, from these training examples. (Row 9, Column 2) depicts the rule surface extracted from this IFT.

The last row of the table shows a sample of the generated rules. This rule states {If Identity is certain, Location is near, Time is prior then Presence_status is online}. The last row also shows the meta-information that complements the rule. This meta-information states that the CMA is the entity who issues these fuzzy rules and applies them to MMARL group and they are valid only to the MMARL lab space.

After building the fuzzy inductive tree, the IA generates the rules that are used in the inference process. The path of each branch from root to leaf is converted into a fuzzy rule with the antecedent part representing the context attribute and its linguistic term of each node along the path, and the consequent part representing the output class at the leaf that has the highest membership value.

The process of inferring new context instances is pursued from the generated rules as explained in Chapter 5. First, each context instance is fuzzified by comparing its captured values with its linguistic terms. Then for each rule, the inference agent selects the minimum value of all the fuzzified contexts in the rule's antecedent part and uses this value as the membership value of the rule's consequent part. The outcome of these steps will be one of the following:

1-All rules have the same consequent. In this case, the IA takes this consequent as the inference result and its membership value equals the minimum membership value of all the rules' consequent output

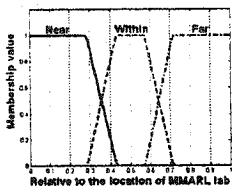
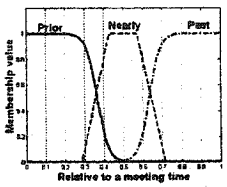
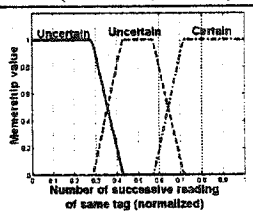
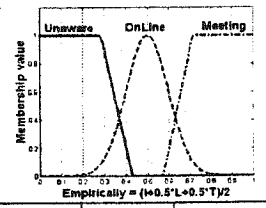
2-Rules have different consequent outputs and with different membership values. In this case, the IA takes the maximum value of all similar consequents, then selects the output consequent with the highest membership

For example, using the tree shown in Table 6-2, let us assume that a fuzzified context instance of a user in the IMS application is equal to {Identity (0.3, 0.7), Location (0.2, 0.7, 0.0), Time (0.5, 0.9, 0.0)}. The output of the generated rules will then be equal to

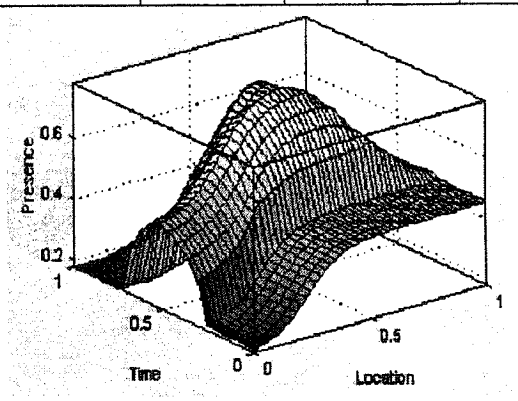
Rule 1: min(0.7,0.2,0.5)= 0.2 → online with MF of 0.2
Rule 2: min(0.7,0.2,0.9)= 0.2 → meeting with MF of 0.2
Rule 3: min(0.7,0.2,0.0)= 0.0→ online with MF of 0.0
Rule 4: min(0.7,0.7,0.5)= 0.5 → online with MF of 0.5
Rule 5: min(0.7,0.7,0.9)= 0.7 → meeting with MF of 0.7
Rule 6: min(0.7,0.7,0.0)= 0.0→ online with MF of 0.0
Rule 7: min(0.7,0.0)= 0.0→ unaware with MF of 0.0
Rule 8: min(0.3)= 0.3 → unaware with MF of 0.3

Since more than one consequent that has the same output class, the IA will take the maximum value of each group of similar consequents, then the maximum of the resultant as follows

$\max(\max(0.2,0.0,0.5,0.0)_{online}, \max(0.2,0.7)_{meeting}, \max(0.0,0.3)_{unaware} = \max(0.5_{online}, 0.7_{meeting}, 0.3_{unaware})=0.7$ → the inference output of that instance has an outcome equal to meeting with membership value equal to 0.7

Location L{near, within, far}			Time T{prior, nearly, past}			Identity I{uncertain, certain}		Presence_status P{unaware, online, meeting}		
										
0.8	0.2	0	0	1	0	1	0	0.6	0.3	0
0	1	0	0	1	0	0.4	0.6	0	0.1	0.8
0	0.3	0.7	0	1	0	1	0	0.7	0.2	0
0	1	0	0	1	0	0	1	0	0.05	1
0	0.8	0.2	0.7	0.4	0	0.2	0.8	0.1	0.9	0
..	..	..	..	..	..	..	..	..	..	..

- **Identity**
  - Certain
    - **Location**
      - Near
        - **Time**
          - Prior → Online
          - Nearly → Meeting
          - Past →Online
      - Within
        - **Time**
          - Prior → Online
          - Nearly → Meeting
          - Past →Online
      - Far→ Unaware
  - Uncertain →Unaware




```
<Obligation rdf:ID="Policy1">
 <name>Group_Presence</name>
 <appliesWhere rdf:resource="&Loc;MMARL"/>
 <enforcementType>
   <PolicyEnforcement rdf:ID="Positive"/>
 </enforcementType>
 <appliesTo rdf:resource="#G1"/>
 <IssuedBy>
   <Confoaf:SoftwareAgent>
     <Confoaf:name>CMA</Confoaf:name>
   </Confoaf:SoftwareAgent>
 </IssuedBy>
 <policyRule rdf:resource="#Frule1"/>
</Obligation>

<Confoaf:Group rdf:ID="G1">
 <Confoaf:name>MMARL</Confoaf:name>
</Confoaf:Group>
```

```
<Fuzz:Min rdf:ID="Frule1">
 <Fuzz:hasAntecedent>
   <Fuzz:Antecedent rdf:ID="Ant1">
     <Fuzz:hasFuzzyVariable>
       <Fuzz:FuzzyVariable>
         <Fuzz:hasName>Identity</Fuzz:hasName>
         <Fuzz:hasFuzzyTerm>
           <Fuzz:FuzzyTerm rdf:ID="Certain"/>
         </Fuzz:hasFuzzyTerm>
       </Fuzz:FuzzyVariable>
     </Fuzz:hasFuzzyVariable>
   </Fuzz:Antecedent>
 </Fuzz:hasAntecedent>
...
</Fuzz:Min>

<Fuzz:hasConsequent>
 <Fuzz:Consequent>
   <Fuzz:hasFuzzyVariable>
     <Fuzz:FuzzyVariable rdf:ID="presence_status">
       <Fuzz:hasFuzzyTerm rdf:resource="#online"/>
     </Fuzz:FuzzyVariable>
   <Fuzz:hasFuzzyVariable>
   </Fuzz:Consequent>
   </Fuzz:hasConsequent>
```

Table 6-2 The inductive fuzzy inference used in the instant messaging application

6.6.4.3 IMS at runtime

At runtime, the IMS combines the inductive fuzzy inference with the ontology-based reasoning in order to cope with the uncertainty in context information and at the same time guarantees that the result of the fuzzy inference is consistent with what users negotiated for and represented as instances of the negotiation ontology.

The first inference action takes place when the CMA receives a *CLS_Negotiation* message. The inference is ontology-type reasoning over the constructs of the negotiation and context ontologies to deduce if the CMA should instantiate a new IMS session for the detected user and to start a negotiation process. This is accomplished by the following rule:

```
[Rule_Start_IMS:
        (?mes, rdf:type, CLNP:NegotiationMessage) ,
        equal (?mes, CLNP:CLS_Negotiate) ,
        (?act, rdf:type, Contivity:Activity) ,
        equal (?act, CLNP:User_Presence) ->
        (CLNP:Start_IMS, rdf:type, Contivity:Activity)]
```

In other words, the rule states: if the received message is of type *CLS_Negotiation* and the specified activity is of type *User_Presence* then start a new IMS session.

The CMA follows by invoking the IA to perform the IFR process. It uses the following rule:

```
[Rule_Engage_IFR:
        (?act, rdf:type, Contivity:Activity) ,
        equal (?act, CLNP:Start_IMS) ,
        equal (?act2, CLNP:User_Presence) ->
        load (MMARL_IA, Presence_Fuzzy_Inference_Tree) ]
```

In other words, if an activity of type *Start_IMS* is generated from the previous rule then call the functor "*load*" that will invoke the *MMARL_IA* inference agent and load the IFR tree shown in the last row of Table 6-2.

After this, the IA starts to use the IFR as discussed in the previous section. However, before applying the results of the fuzzy induction, the CMA must guarantee that those

results comply with the negotiated context information. This is accomplished by using the IFR results to trigger the negotiation rules. For example, in section 6.6.4.1, the user specified that his presence is not to be projected to users in the *MIRLGroup*. This rule is transformed to the following negotiation rule

```
[Rule_Apply_CLNP_Rule:
        (?clnp, CLNP:hasRequester, ?requester), (?clnp, CLNP:hasProcess, ?mes),
        (?mes, rdf:type, CLNP:NegotiationMessage), (?mes, CLNP:hasRule, ?rule),
        (?antc, ConRule:relation, ConPolicy:Negative), (?antc, ConRule:entityB, ?group),
        (?group, rdf:type, Confoaf:Group), (?person, rdf:type, Confoaf:UA),
        (?person, Confoaf:hasGroup, ?Group) ->
        (?requester, Confoaf:hasActivity, Contivity:offline),
        changeStatus (?requester, ?person)
```

In other words, if the IMS has a *requester* that specifies his presence to *MIRLGroup* is *Negative*, then the *requester's* IMS status will be *offline* and the CMA invokes the functor "*changeStatus*" to apply this action.

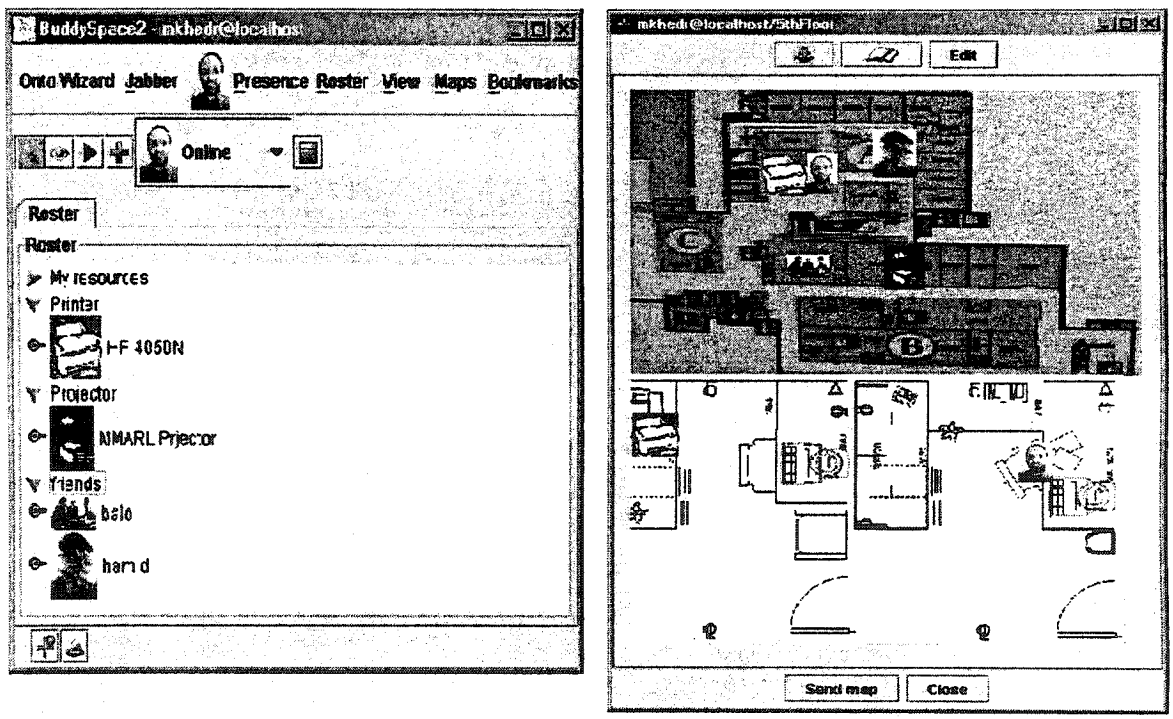


Figure 6-18 IMS Interface
a) Snapshot of the IMS user interface. b) Map with entities, their locations and activities.

Figure 6-18a shows the buddy-list of user *khedr* and the services that can be used by this user, the HP printer and the MMARL projector. These services are projected on his buddy-list according to his location, activities, time and generated fuzzy rules. The figure also shows that the projector is currently busy, as its icon is red, and that his friend *Balo* is in a meeting.

Figure 6-18a also shows *Hamid* having a blurred icon in user *khedr's* buddy-list. This is an indication that ACAI is un-aware of the presence status of *Hamid.* It means that *Hamid* could be present in the environment but his presence is vague.

Figure 6-18b is a map service in the IMS that projects the activities and locations of entities found in the user's buddy-list. It is a two layer map, where the above layer is the map of the fifth floor in Colonel-By building while the lower is the map of *Khedr's* office in that floor. Every user in the buddy-list is projected to the map so that *Khedr* can have knowledge about their location spontaneously as they move in the fifth floor.

Finally, every user can manipulate his ACF, CIF and other ontology profiles through the OntoWizard menu item found in the toolbar of the IMS GUI. This OntoWizard menu also allows users to browse and search the system repository if they have the right permissions to do so.

6.7 External Evaluators

By the time of writing this thesis, two researchers in our lab (besides the author) are using ACAI infrastructure in their research to develop applications for mobile applications and to provide security and session management based on context and policies.

They are using the developed fuzzy ontology for defining policies and extending the context ontologies to include security support. Also, the negotiation process is used by one of the researchers to adapt his proposed framework for mobile environment according to the semantically represented context [24],[57] ,[68].

This section summarizes their critical review about ACAI and if they considered ACAI's components to be useful. The section is divided into four subsections; each corresponds to a question given to the developers.

6.7.1 Easiness of Application Development

The goal of the question was to understand how easy or difficult it was to get used to ACAI APIs to develop their applications. The developers agreed that ACAI's API was quite adequate for the type of applications they built, and was easy to get used to it. One of them mentioned that reusing the fuzzy ontology to define policies was straightforward to understand, reuse, and extend.

6.7.2 Applicability of ACAI Components

ACAI components are designed to support the development of possibly any type of context-aware applications. Therefore, it has to be generic enough to address different types of applications while providing enough functionality to make it useful.

According to the developers' opinion, the developed ontologies supported and simplified the development of their applications, and they did not require any changes in their original designs to accommodate their applications to ACAI ontologies. However, when developing a security-related application, the developer said that the fuzzy ontology did not provide advantage to his application. The reason is that it is not realistic to fuzzify security-related information, as it will open the door for leakage of information and security threats. However, our discussion resulted in that this issue can be resolved by using crisp membership functions for security-related information, since this will turn fuzzy logic to ordinary logic, allowing him to avoid the potential security problems.

6.7.3 Complexity of the ACAI's Components

ACAI is built on the concept of profiling context and system information using ontologies. Both application developers agreed that learning the ontology was time consuming and one of the developers pointed out that learning ACAI's inductive fuzzy inference API to leverage its functionality required a significant amount of time.

We have already addressed some of these issues. For example, ACAI now provides GUI interfaces for writing and editing ontologies. These GUIs allow developers to specify their contextual information without going into OWL syntax and structuring.

6.7.4 Adaptation Capability

Both developers agreed that the negotiation mechanism was useful in developing applications that require the support of runtime adaptation. The negotiation ontology and decision-making that is based on utility functions were the two most appreciated aspects in the negotiation. The developers pointed out that the ability to negotiate context information simplified the coding of their mobile applications since mobility was managed according to this negotiated context.

Based on the reviews of the developers, we can conclude that ACAI provides useful functionalities to develop context-aware applications. Most of the problems encountered during the application development process were related to bugs, poor documentation of ACAI's API, complexity of using ontologies, and lack of a mechanism to automate the fuzzy membership function elicitation. As more applications are built, we expect to simplify the application development process even further.

6.8 ACAI Performance

The goal of this thesis is to provide computational support for the development and management of context-aware applications in pervasive environments. To support this thesis, I and other developers in the MMARL lab developed applications targeting different areas in the pervasive computing field. I showed in the previous sections some of these applications and discussed the merits that ACAI helped in to support these applications.

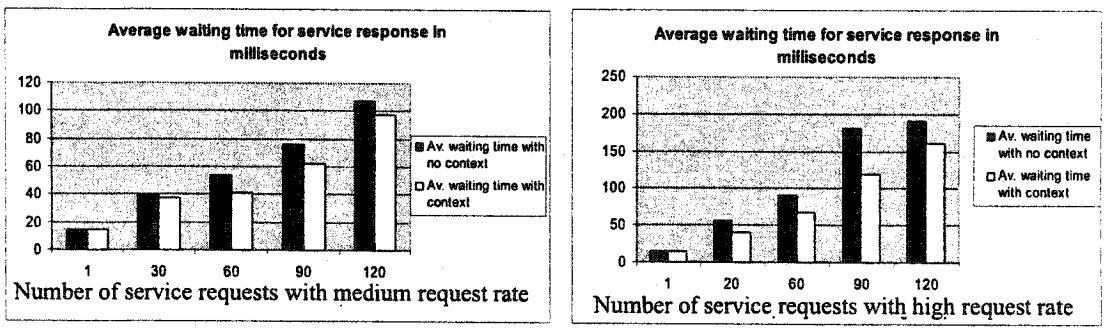
In this section we discuss the performance evaluation of ACAI to validate the thesis that our infrastructure components require insignificant time to respond to service requests,

linearly proportional to these requests, and that the overhead imposed by the negotiation is justified with the increase in the performance of the system.

In order to validate this hypothesis, we have implemented a benchmark and used it to run a large number of experiments. The benchmark allows us to customize both: the ontology profiles used in the experiments and negotiation parameters. We can decide the number of users, number of service requests, number of context providers, types of context, types of service requests, type of inference used, parameter values used in the inductive fuzzy reasoning and parameters of the utility function used in the negotiation process.

All tests were performed on Toshiba laptops equipped with 128 MB RAM, Intel Pentium II processor of 1 GHz and connected to WLAN network using Lucent 10Mbps wireless cards. The operating system is Windows XP, the Java virtual machine version is 1.4.1 and we used JADE version 3.01 as the agent platform.

6.8.1 Impact of Incorporating Context



a b

Figure 6-19 Average waiting time for service provisioning

The first set of experiments measures the impact of using context information to provide services to entities in an ACAI environment. Typical applications used in these experiments are the printing service that I presented in section 6.6.2 and instant messaging discussed in section 6.6.4.

Context information used in these experiments varies according to the type of applications used in the evaluation. Thus, we compare in the evaluation between either having context information taken into consideration while providing a service or context information not considered at all. Experiments were averaged over the number of times

they were repeated to have a reasonable review about the values we calculated. Most experiments were repeated between 20 and 30 times using the same computing capabilities but randomly changing the types of service requests, and the contextual information used in these requests.

Figure 6-19 shows the encountered average waiting time in milliseconds in order to project services that match user's requests to the user. In the conducted experiments, this depends on two factors: service availability and average service time.

We first conducted the experiments with no context information used in the process of service provisioning. We then repeated the experiments taking into account context information, such as location (users and services), utilization of the current resources and agent activities. We found that performance was much better in the second case. This is because contextual information was used in the lookup of services that match users' requests and in deciding the rules to execute to govern the service provisioning process.

One can observe from Figure 6-19a and b, that at small number of requests, using context did not reduce the average waiting time significantly. Nevertheless, as the number of requests increases and their rate of arrival to the CMA increases, using context information reduced the average waiting time to respond to users' requests. In fact, as shown in Figure 6-19a with a moderate request rate, using context information reduced the average waiting time by nearly 20% and with high request rate, using context reduced the average waiting time by nearly 25% as shown in Figure 6-19b.

However, with large number of requests and high rate of arrival, the impact of using context tends to decrease but is still significantly above the no-context situation. This is observed in the case of 120 requests in Figure 6-19b. The reason behind this is that at high requests rate, ACAI spends more time on inferring, matching, and enforcing rules to decide the best services that suit the requests. We consider this an advantage in our system, rather than a disadvantage, as the system will provide a better matching service even with high requests rate. Figure 6-20 illustrates this conclusion.

Figure 6-20 shows the percentage of successful service provisioning (matching and executing) to user requests, like providing a printer that matches user's context and

requirements, or projecting user's presence in the instant messaging service without violating user's privacy information.

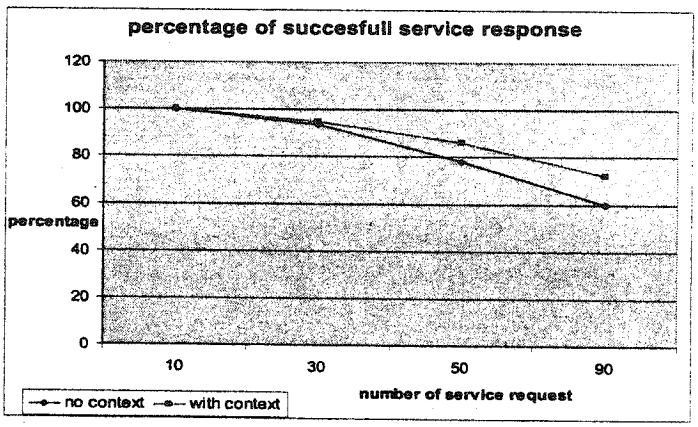
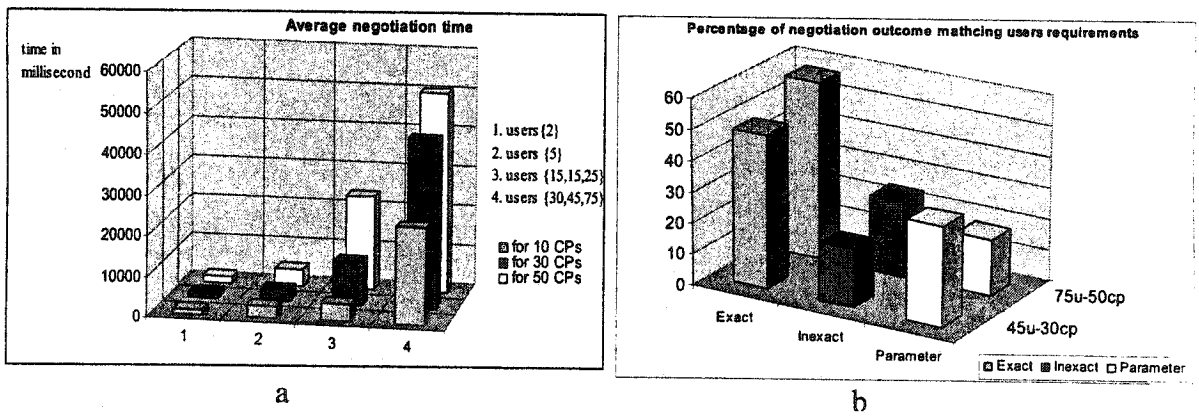


Figure 6-20 Percentage of successful service matching and execution

Figure 6-20 proves that at small number of requests, using context information is not of significant in the process of service provisioning since services are available and free to invoke. However, with the increase in number of requests, ACAI with its context-awareness capability outperforms the case where context is not used. A gain of at least 12% is reported when the number of requests is in the range of 90 requests and with high request rate in the range of 1 request per second.

6.8.2 Impact of Negotiating Context Information



a b

Figure 6-21 Impact of negotiating context information

Negotiating context information is a novel technique to adapt the behaviour of context-aware systems in general and ACAI in particular. However, negotiation will impose

overhead to the infrastructure operation. This is because of the additional messages exchanged among agents to set context information used in the application, such as the event-notification application, and the time delay encountered until the negotiation is finished.

This section reports our concluded remarks on the performance of ACAI when negotiation is used to set context information for system adaptation. The carried experiments measured the average time to complete a negotiation, the average number of messages exchanged in the negotiation, and the percentage of successful negotiated outcomes that matched user requirements. Experiments were conducted by varying the number of users, number of context providers, and types of context information used in the negotiation as well as the range of these contexts.

Figure 6-21a is a comparison among different experiments that measure the average negotiation time to reach an agreement when the number of users and the number of CPAs involved in the negotiation are varied. We used three types of contextual information in these experiments and varied their ranges randomly each time we repeat the experiments. The figure shows that the average negotiation time is nearly linearly proportional with the number of negotiation requests, except for the last values, which is a good indication of the system scalability.

The reason for the increase in the average negotiation time in the last values of Figure 6-21a is attributed to the fact that the number of negotiation requests is at least one and half the number of context providers and thus ACAI took a longer time to set an agreement between the context providers and the users. However, this also indicates that our system is gracefully degrading in performance when it is highly overloaded.

After closely inspecting the negotiation process, we found that the main contribution to the increase in the average negotiation time was the inference process and matchmaking and not the negotiation mechanism itself. The number of exchanged messages among entities involved in the negotiation, as shown in Figure 6-22, proves this conclusion.

Figure 6-22 shows that the increase in the average number of messages has an upper limit of about eight messages. This corresponds to two rounds in the negotiation process. Thus, the time taken to negotiate context to increase system adaptability is highly dependent on

the inference and matchmaking and not on the negotiation mechanism. In my future work, I will investigate if the inference implementation can be enhanced to fast the inference process or I might investigate new trends in fuzzy inference such as the possibility reasoning [59],[58].

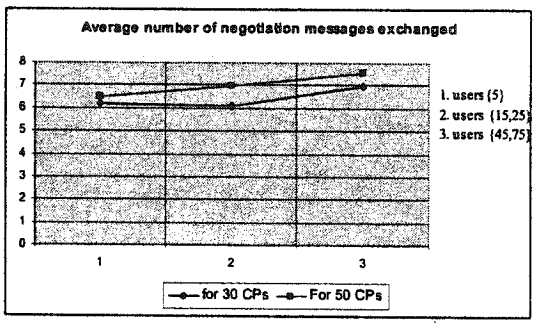


Figure 6-22 Average number of messages exchanged in the negotiation

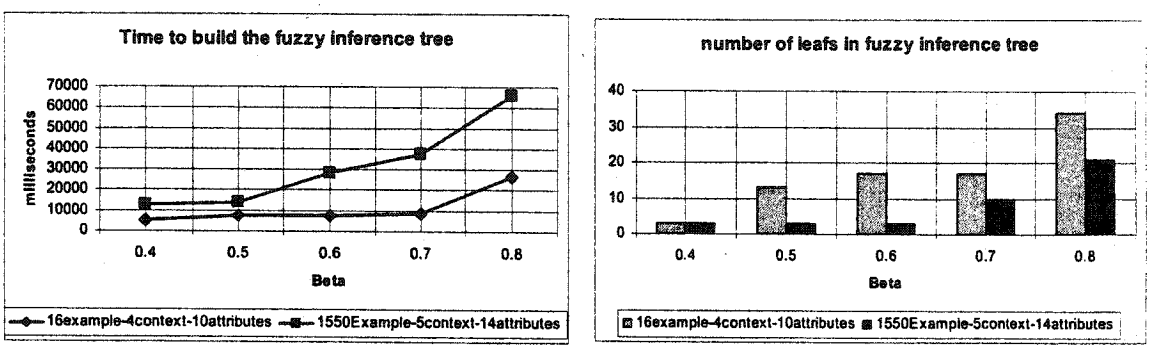
However, this increase in the negotiation delay is justified with Figure 6-21b. This figure shows the percentage of successful negotiation outcomes that are categorized as exact matching, inexact matching, and parameter matching. As the figure illustrates, the percentage of exact-matching is higher in case of having larger number of context providers in the system, even with the increase in the number of users. This is because the inference agent will have more alternatives to choose from, compare among, and to decide accordingly.

It is noteworthy that we conducted the experiments using one CMA, which we think contributed to the increase in negotiation delay. More experiments in the future will reveal the impact of using several CMA agents and the overhead that will occur from coordinating these context management agents.

6.8.3 Impact of Fuzzy Inference

The next type of experiments measures the effect of using fuzzy logic and inductive fuzzy inference to reason about context in ACAI environment. The conducted experiments measure the time taken to build the fuzzy inference tree, the size of the tree and the percentage of acceptable inference outcomes when varying the cut-off parameter (β) used in building the inference tree.

The experiments where carried on two sets of examples. The first has a relatively small number of examples, 16 examples, while the other set contains large number of examples, 1550 examples. In addition, we used four types of context information with 10 linguistic attributes in the first case, while we used five types of context information with 14 linguistic terms in the second case. Figure 6-23 depicts the obtained results.



a b

Figure 6-23 Impact of fuzzy inference

Figure 6-23a illustrates the time taken by the inference agent to build the fuzzy inference trees of these two sets. Note that this process is only required once, during the setup of ACAI, and can be repeated after long periods of time to update the generated tree with the new entries that are added to the example space.

The results show that for the small number of examples, varying β has no significant effect on either the time to build the tree or the size of the tree as shown in Figure 6-23a and in Figure 6-23b respectively. However, with high β values, the time to build the tree increases as in the case of β=0.8. This is because the inference agent tries to learn the pattern of the data and thus it increases the size of the generated tree, which contributes to the increase in the time needed to build the inference tree.

An interesting observation is that in the case of the small set of examples, the accuracy of the fuzzy inference, shown in Figure 6-24, does not increase significantly with the increase of β values. This is because the inference agent memorizes the examples and does not generalize them in order to increase its inference capability.

This is not the case when it comes to the large set of examples as illustrated in Figure 6-23. In this case, increasing β has the effect of increasing the size of the tree and the time

taken to build it. However, the inference agent does not memorize the context pattern as in the case of the small set of examples, and thus provides better inference with higher β values as illustrated in Figure 6-24.

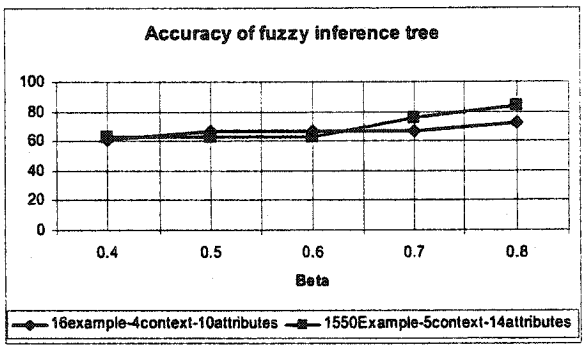


Figure 6-24 Accuracy of the inductive fuzzy inference

From these experiments, we conclude that choosing the size of the set of examples is highly dependent on the situation where ACAI is implemented. In case like the event-notification application, small set of examples is better since the number of contexts used is small, mainly location, time and the activity, and at the same time, the process of inference is required to be fast.

In the case like instant messaging application, where the types of context used are quite large, using large set of examples is better and will provide more accurate inference outcome, which is more crucial than the speed of the inference process.

6.8.4 Overall Performance Discussion

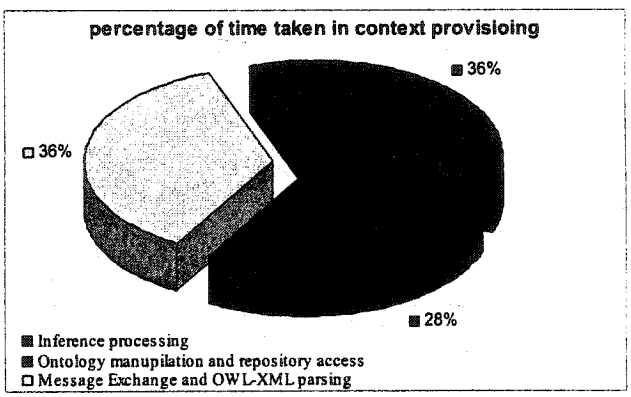


Figure 6-25 Percentage of time required for context provisioning

The process of evaluating the performance and soundness of ACAI is not quite straightforward since its components are tightly integrated and isolating each component to test its effectiveness is difficult to accomplish. Nevertheless, in the previous sections I reported the evaluation of each component individually as much as possible.

This section discusses the overall evaluation of ACAI in terms of the time taken to perform a complete context-provisioning life cycle. In other words, the time taken to accomplish a task that consists of parsing an incoming request or event, accessing the system repository for information and service lookup, matching and inferring over the results and responding to the received request or event. The reported evaluation is based on monitoring ACAI's behaviour when running the aforementioned applications for several times and in different situations.

Figure 6-25 illustrates the time taken by ACAI, as a percentage, for accomplishing the context provisioning life cycle. ACAI nearly consumes 36 % of the time in the inference process. This includes the fuzzification, inference, and decision-making steps. It consumes another 36% of the time to parse and exchange OWL format messages among the agents involved in this task. The remaining 28% is used to access the system repository and manipulate the ontology information.

These evaluations emphasizes the advantage of incorporating ontologies, negotiation and IFR in context-aware environments since their introduction although cause on average an increase of 36% in the context provisioning time, they also increase the number of successful and exact service provisioning by 12%. In numbers, the average time to complete a context provisioning cycle in the IMS is about 40 seconds, with negotiation is about 55 seconds, on the other side with 90 requests and one second arrival rate, 54 exact responds were delivered without negotiation and 72 exact responds in case of negotiation.

Time might be crucial in certain applications such as emergencies. That's why we are investigating enhancement methods in the inference algorithm to reduce the context provisioning time in future versions of ACAI

6.9 Summary

This chapter presented the applications implemented using ACAI infrastructure and discussed the results of its evaluation. The conference application and its associated sub-applications have been used to discuss the qualitative results, both in terms of usability and effectiveness. This was followed by a thorough evaluation of ACAI performance with respect to every aspect we introduced and discussed in this thesis.

In terms of qualitative results, we have shown that ACAI offers suitable abstractions that ease the development of context-aware applications in pervasive environments. Application developers do not have to deal with the heterogeneity of context sources or to query directly their status. The task they are required to undertake is to gather information about the applications they are developing in the form of users' and services' preferences as well as the environment contexts requirements, and to process this information to allow for the use of the fuzzy inference and the negotiation mechanism.

As for performance, we discussed the different evaluations that we conducted and reported the values we measured for every aspect implemented in the current version of ACAI as well as an overall evaluation of ACAI operation.

Chapter 7

Conclusions and Future Work

The main goal of the work presented in this thesis has been the development of abstractions and algorithms that, embedded in a pervasive computing infrastructure, effectively enhance the development and management of context-aware applications. More precisely, I have designed a generic infrastructure to support and facilitate the development of context-aware applications in pervasive environments. This was followed by an in depth modeling of context information using ontologies to achieve common understanding of context and to support reasoning about context. This modeling approach lead to a set of ontologies that contain new ontologies as well as extended versions of existing ones to cope with the dynamic nature of pervasive environments.

Our work was the first to investigate and use semantically modeled fuzzy logic to reason about uncertainty in captured context and this approach lead to the development of a novel algorithm for reasoning about context in pervasive environments. Finally, I developed a negotiation mechanism to define and set contextual information that are relevant to users and entities in the environment. This led to a more reliable and adaptable infrastructure.

I discussed the implementation of ACAI and applications that were developed on top of it to demonstrate its effectiveness in developing context-aware applications. I provided an evaluation of the different components of the infrastructure and a critical review about its overall performance.

In this chapter, I revise the main contributions of the thesis, provide a critical evaluation of the goals attained and at the end of the chapter, I will discuss some open issues that I leave for future development.

7.1 Contributions

The contributions of this thesis targeted the field of pervasive environments and specifically context-awareness. These contributions are summarized as follows.

7.1.1 Semantic Modeling of Context

Our work contributes to the field of context-awareness with the development of a set of ontologies that enables the semantic modeling of context and meta-context information.

This approach of semantically modeling context using ontologies provides the proposed infrastructure with sound characteristics such as

- Interoperability: ontologies provide common understanding of the terminologies used in the environment.
- Reusability: ontologies provide the required functionalities to be reused and implemented in different applications.
- Extensibility: ontologies support the process of extensibility. This can be achieved using the import mechanism or locally modifying original ontologies.
- Reasoning: ontologies have the intrinsic capability of providing reasoning mechanism such as subsumption, equivalence, restriction, and consistency.

In addition, our approach to model the theory of fuzzy logic and the negotiation mechanism in the form of ontologies enriched ACAI with the following characteristics:

- Separation of concerns: by semantically modeling the meta-information of the infrastructure, developers can freely modify parameters and preferences of the infrastructure without significant changes in the implementation of the infrastructure.

- Robustness: the process of semantically representing the negotiation process and its messages allows for the runtime execution of the negotiation protocol that best suits the current situation of agents involved in the application. In addition, agents can modify their decision mechanism without changing their coding behavior.

7.1.2 Fuzzy Logic for Reasoning about Uncertainty in Context

Our work contributes to the field of context-awareness with a mechanism for inductively reasoning about context information in uncertain and vague situations using fuzzy logic. Our work was the first to investigate the process of fuzzifing semantically modeled information in general and context in specific. The process of fuzzy inference included:

- Fuzzifing context: this process uses the designed fuzzy ontology to define membership functions for semantically modeled contextual information used in applications running over ACAI. This includes the algorithm used to fuzzify context and performs membership elicitation.
- Building an inductive fuzzy tree: an entropy-based algorithm for building an inductive fuzzy tree was introduced. This tree is used to generate fuzzy rules that specify the conditions needed to be satisfied in order to perform actions or trigger events.
- The inference process: this process stated the steps taken to reason about captured contexts using the rules generated from the inductive fuzzy tree and methods of resolving conflicts.

The generated fuzzy rules are used along with the process of negotiating context to set context levels required by applications and entities using ACAI.

7.1.3 Adaptation using Context Negotiation

The proposed negotiation mechanism for specifying and setting contexts that are relevant to the current situation is our contribution to the adaptation axis in the three-axis of research interests that I explained in Chapter 2 and illustrated in Figure 2-1.

Adapting the infrastructure behaviour using negotiation has the following unique advantages:

- Provides a declarative method for users to state their context requirements, and for context providers to offer dynamic customized context-based service information.
- Protects context providers against unexpected degradation in performance. Since they know users' expected requirements, they can better manage their resources.
- Provides an assurance of seamless context-based service provisioning in large environments.
- Provides an easy mechanism for context providers to intelligently supply and monitor contextual resources: they can obtain and analyze user data without unnecessary distractions for the user.

As explained in Chapter 3, context negotiation is carried out using the ontologies designed to represent context and based on a multi-attribute logarithmic utility function for negotiation evaluation and decision-making.

This approach differentiates my work from existing approaches that utilize scripting techniques to specify when, how and what to do to adapt to the system behaviour. The scripting technique has the advantage of simplicity and ease of implementation but on the other side it lacks the sound capability of dynamically allowing the system to adapt to changes in the environment at runtime.

7.2 Evaluation

Evaluating pervasive environments in general and context-aware environments in specific are extremely hard. This is because there are no agreed upon mechanisms or measurements that any developer can follow in order to evaluate his system neither in the form of comparison with other systems nor as a standalone tests to measurement the effectiveness and applicability of the system.

Most projects try to prove their system goodness by implementing different applications targeting diverse situations in pervasive environments and argue that this is an evidence

of their system effectiveness. Other projects did try to define parameters and measuring methods to evaluate their system such as response time and system overloading performance.

My thesis is no exception when it comes to its evaluation. I took the same procedure of implementing different applications as proof of concept of the infrastructure applicability and I measured its performance and system behaviour in extreme situations as a proof of its scalability, effectiveness, and good performance.

7.3 Building Applications and Infrastructure Evaluation

The infrastructure we have developed specifically targets pervasive environments, i.e. environments with dense, capability-limited, and portable resources. This placed constraints on the functionalities that can be supported by the infrastructure. Specifically, the infrastructure has to be interoperable, lightweight, and with minimum overhead.

In Chapter 6, I showed that ACAI can support pervasive environments and that it met these requirements.

The semantic modelling of both context and inference process facilitated the interoperability of ACAI. New applications as well as existing ones can be easily composed, integrated, and interacted as they all have the same semantic knowledge of the information and surroundings they use.

Negotiating context information was also proven to be a sound proposition as it elevated the interaction between the two ends of the system, (consumers and providers), from the static level to the dynamic level. Negotiating context allowed both sides to expose and share their intended means, requirements, and provisions in terms of context information. This increased the system performance as I showed in Chapter 6 and at the same time reduced system independency of consumers' situations.

7.4 Future Work

The work presented in this thesis establishes a concrete foundation of an infrastructure and its components for enhancing the development and management of context-aware applications in pervasive environments.

Based on this foundation, future developments include the following.

- **Improvements to the Current Infrastructure Architecture**

Although our proposed infrastructure is designed to be generic enough to support wide range of context-aware applications, there are areas that my thesis did not investigate such as inter-communications between the components of the infrastructure and how this communication can be context-sensitive. By context-sensitive, we mean the ability to detect variation in context and not context itself. We have the instinct that this will greatly reduce the development of context applications not only on the medium scale as proposed in my thesis but on the large scale and even on an Internet scale.

We did not investigate security issues in our infrastructure. Security has important aspects that need to be addressed for the success of context-aware environments. This is because users will not be willing to sacrifice their privacy and personal information for the sake of value-added services in their environments.

- **Revise the Semantic Modeling of Context**

No one can claim that his developed ontologies are the best in their domains. It always gets down to the ability to use these ontologies in useful applications. We showed that our ontologies are in fact good enough to model a wide range of context-aware applications. What we need to further investigate as a future work is how to come up with some standard process that can prove, to some extend, the effectiveness of ontologies

- **Enhance Uncertain Reasoning**

The fuzzy reasoning process and the algorithms developed in this thesis to reason about context awareness facilitated the automation needed to take decision, trigger actions, deduce higher context information, and increase reliability. I will investigate in the

coming future how this fuzzy reasoning can be accompanied by probabilistic/possibility methods such as Bayesian networks [69],[54],[55] and what the pros and cons of performing this combination are. In addition, methods of self-generation of membership functions and tuning of generated inference trees are in the list of my future work.

- **Investigate other Methods in the Negotiation Process**

We believe that the negotiation of context information is a breakthrough in the field of context-awareness since it enables the runtime and dynamic adaptation of the system behaviour to users' requirements that are based on context information.

As future work, I will investigate the ability to use different utility functions and may come up with a model that can decide the best utility function to use for every context captured in the environment. This is quite hard as context is still in its infancy phase and researchers are still struggling with the fact of what context is.

References

[1] Mark Weiser, "The Computer for the Twenty-First Century," Scientific American, September 1991, pp. 94-10.

[2] Mark Weiser, "Hot Topics: Ubiquitous Computing," IEEE Computer, Vol. 26, No. 10, October 1993, pp. 71-72.

[3] N. Davies, H. Gellersen, "Beyond Prototypes: Challenges in Deploying Ubiquitous Systems," IEEE Pervasive Computing, Vol. 1, No.1, January-March 2002, pp. 26-35.

[4] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," IEEE Personal Communications, Vol. 8, No. 4, August 2001, pp. 10-17.

[5] Brumitt, B., et al., "Ubiquitous Computing and the Role of Geometry," IEEE Personal Communications, Vol. 7, No. 5, August 2000, pp. 41-43.

[6] D. Saha, A. Mukherjee, "Pervasive Computing: A Paradigm for the 21th Century," IEEE Computer, Vol. 36, No. 3, March 2003, pp. 25-31.

[7] C. Mascolo, L. Capra, W. Emmerich, "Middleware for Mobile Computing," Networking 2002, Tutorial Papers, 2002.

[8] S. Helal, "Standards for service discovery and delivery," IEEE Pervasive Computing, Vol. 1, No. 3, July-Sept. 2002, pp. 95–100.

[9] G. Chen, D. Kotz, "A Survey of Context-Aware Mobile Computing Research," Technical Report, TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[10] Nigel Shadbolt, "Ambient Intelligence," IEEE Intelligent systems, Vol. 18, No. 4, July-August 2003, pp. 2-3.

[11] A. dey, et al. "Enabling the Use of Context in Interactive Applications," CHI2000 Doctoral Consortium, April 2000, pp. 79-80.

[12] A. Schmidt, M. Beigl, H. W. Gellersen "There is more to context than location," Computers & Graphics Journal, Elsevier, Vol. 23, No. 6, Dec 1999, pp. 893-901.

[13] A. dey, et al., "A Context-based Infrastructure for Smart Environments," 1st International Workshop on Managing Interactions in Smart Environments, December 1999, pp. 114-128.

[14] Anand Ranganathan, Roy H. Campbell, "A Middleware for Context-Aware Agents in Ubiquitous Computing Environments," International Middleware Conference, Rio de Janeiro, Brazil, 2003, pp. 143-161.

[15] M. Khedr, A. Karmouch, "ACAI: Agent-Based Context-aware Infrastructure for Spontaneous Applications," Journal of Network & Computer Applications, to appear.

[16] OWL Web Language Ontology Reference. http://www.w3.org/TR/owl-ref.

[17] The Foundation of the Intelligent Physical Agents. http://fipa.org

[18] Stuart J. Russell and Peter Norvig, "Artificial intelligence: a modern approach," Upper Saddle River, N.J. Prentice Hall, 2003.

[19] J F Baldwin, J Lawry, T P Martin "A Mass Assignment Based ID3 Algorithm for Decision Tree Induction," International Journal of Intelligent Systems Vol. 12, No. 7, 1997, pp. 523-552.

[20] M. khedr, A. Karmouch, "Exploiting SIP and agents for smart context level agreements," IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, Victoria, BC, Canada, August 2003, pp. 522-525.

[21] M. Khedr, A. Karmouch, R. Liscano, T. Gray, "Agent-based Context Aware Ad hoc Communication," MATA'02, October 2002, pp. 105-118.

[22] M. Khedr, A. Karmouch, "A Semantic Approach for Negotiating Context Information in Context Aware Systems," IEEE Intelligent Systems Magazine, to appear.

[23] M. Khedr, A. Karmouch, "Customizing Pervasive Environments using Context Level Agreements," IEEE Pervasive Computing, under second review.

[24] H. Harroud, M. Khedr, A. Karmouch, "Building Policy-based Context Aware Applications for Mobile Environments," IEEE/IFIP Mobility Aware Technologies & Applications, Oct. 20-22, 2004, Brazil.

[25] A. Karmouch, A. Galis, R. Giaffreda, T. Kanter, et al., "Context Aware Research Challenges in Ambient Networks," IEEE/IFIP Mobility Aware Technologies & Applications, Oct. 20-22, 2004, Brazil.

[26] M. Khedr, A. Karmouch "A Context-Sensitive Middleware for Managing Embedded Pervasive Environments," The 2004 International Conference on Embedded And Ubiquitous Computing, Japan, August 2004, pp. 1085-1095.

[27] A. K. Dey, G. D. Abowd, "Towards a Better Understanding of Context and Context-Awareness," Proc. of CHI, the Netherlands, April 2000.

[28] Vaughan-Nichols, S.J.; "Wireless middleware: Glue for the mobile infrastructure," IEEE Computer ,Vol. 37, No. 5, May 2004, pp.18 – 20.

[29] Teller, S.; Jiawen Chen; Balakrishnan, H. "Pervasive pose-aware applications and infrastructure," Computer Graphics and Applications, IEEE, Vol. 23, No. 4, July-Aug. 2003 pp.14 – 18.

[30] Schulzrinne, H.; Xiaotao Wu; Sidiroglou, S.; Berger, S.; "Ubiquitous computing in home networks," Communications Magazine IEEE, Vol. 41, No. 11, Nov. 2003, pp. 128 – 135.

[31] Brewster, C.; et al.; "Knowledge representation with ontologies: the present and future," Intelligent Systems, IEEE, Vol. 19, No. 1, Jan.-Feb. 2004, pp. 72-81.

[32] Gauvin, M.; Boury-Brisset, A.-C.; Auger, A., "Context,ontology and portfolio: key concepts for a situational awareness knowledge portal System Sciences," Proceedings of the 37th Annual Hawaii International Conference on, 5-8 Jan. 2004.

[33] Beneventano, D.; Bergamaschi, S.; Guerra, F.; Vincini, M.; "Synthesizing an integrated ontology," Internet Computing, IEEE , Vol. 7, No. 5, Sept.-Oct. 2003, pp.42 – 51.

[34] Tse-Ming Tsai, et al, "Ontology-mediated integration of intranet Web services," IEEE Computer, Vol. 36, No. 10, Oct. 2003, pp.63 – 71.

[35] Masuoka, R.; Labrou, Y.; Parsia, B.; Sirin, E.; "Ontology-enabled pervasive computing applications," Intelligent Systems, IEEE, Vol. 18, No. 5, Sep-Oct 2003, pp.68 – 72.

[36] Vasiu, L.; Mahmoud, Q.H., "Mobile agents in wireless devices," IEEE Computer, Vol. 37, No. 2, Feb. 2004, pp.104 – 105.

[37] Stroulia, E.; Hatch, M.P., "An intelligent-agent architecture for flexible service integration on the web," Systems, Man and Cybernetics, Part C, IEEE Transactions on, Vol. 33, No. 4, Nov. 2003, pp. 468 – 479.

[38] Gunter, M.; Braun, T.; "Internet service monitoring with mobile agents," IEEE Network, Vol. 16, No. 3, May-June 2002, pp. 22 – 29.

[39] Korpipaa, P.; et al.; "Managing context information in mobile devices," Pervasive Computing, IEEE, Vol. 2, No. 3, July-Sept. 2003, pp. 42 – 51.

[40] Schilit, B.N.; Hilbert, D.M.; Trevor, J.; "Context-aware communication," IEEE Wireless Communications, Vol. 9, No. 5, Oct. 2002, pp. 46-54.

[41] Voida, S.; et al; "Integrating virtual and physical context to support knowledge workers," Pervasive Computing, IEEE , Vol. 1, No. 3, July-Sept. 2002, pp.73 – 79.

[42] Bellavista, P.; Corradi, A.; Stefanelli, C.; "The ubiquitous provisioning of internet services to portable devices," IEEE Pervasive Computing, Vol. 1, No. 3, July-Sept. 2002, pp. 81-87.

[43] Garlan, D.; et al; "Project Aura: toward distraction-free pervasive computing," Pervasive Computing, IEEE, Vol. 1, No. 2, April-June 2002 , pp. 22 – 31.

[44] Berners-Lee, Tim, Hendler, James, and Lassila, Ora, "The Semantic Web," Scientific American, Vol. 284, No. 5, pp. 35-43, 2001.

[45] Fensel Dieter, "Ontology-Based Knowledge Management," IEEE Computer, Vol. 35, No. 11, 2002, pp. 56-59.

[46] Fensel Dieter, "Ontologies: a silver bullet for knowledge management and electronic commerce," Berlin, Springer, 2001.

[47] Ontology Development 101: A Guide to Creating Your First Ontology. http://protege.stanford.edu/publications/ontology_development/ontology101.html

[48] The Context Ontology. http://www.schemaweb.info/schema/SchemaDetails.aspx?id=187

[49] The Time Ontology. http://www.cs.rochester.edu/~ferguson/daml/

[50] Friend Of Friend Ontology. http://xmlns.com/foaf/0.1/

[51] Java Agent Development Framework. http://jade.tilab.com/

[52] Jeremy J., et al, "Jena: Implementing the Semantic Web Recommendations," Technical report, HPL-2003-146, HP Laboratories, Bristol.

[53] Craig Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process," Prentice Hall, 2001.

[54] Zhongli Ding, Yun Peng. "A Probabilistic Extension to Ontology Language OWL," 37th Hawaii International Conference On System Sciences (HICSS-37), January, 2004, Hawaii.

[55] P. Korpipaa, et al, "Bayesian Approach to Sensor-based Context Awareness," Personal and Ubiquitous Computing Journal, Vol. 7, No. 2, 2003, pp. 113-124.

[56] http://buddyspace.org

[57] Ganesan, P., Karmouch, A. "Context Awareness in Ad hoc Communications," IEEE PACRIM, August 2003, pp. 919-922.

[58] Zadeh, Lotfi, "Fuzzy Sets as the Basis for a Theory of Possibility," Fuzzy Sets and Systems, Vol. 1, No. 1, 1978, pp. 3-28.

[59] D. Dubois, et al, "Possibility Theory," Plenum Press, New York, 1988.

[60] Hazas, M.; Scott, J.; Krumm, J.; "Location-aware computing comes of age," IEEE Computer, Vol. 37, No. 2, Feb. 2004, pp. 95 – 97.

[61] Anind Dey, "Understanding and using context," Personal and ubiquitous computing, Vol. 5 , No 1, 2001, pp 4-7.

[62] Daniela Petrelli, et al, "Modeling and Adapting to Context," Personal and ubiquitous computing, Vol. 5, No.1, 2001, pp 20-24.

[63] A. Schmidt, H. W. Gellersen, and M. Beigl., "A Wearable Context-Awareness Component - Finally a Good Reason to Wear a Tie," Proceedings of the third International Symposium on Wearable Computers. San Fransico, 18-19. Oct. 1999. pp 176-177.

[64] A. Schmidt, H.W. Gellersen, and M. Beigl, "Matching Information and Ambient Media," In Proceedings of CoBuild'99. Second International Workshop on Cooperative Buildings, Pittsburgh. Springer: Heidelberg 1999.

[65] A. Schmidt, et al., "Advanced Interaction in Context," 1th International Symposium on Handheld and Ubiquitous Computing (HUC99), Germany, 1999, pp 89-101.

[66] A. Schmidt and J. Forbess, "What GPS Doesn't Tell You: Determining One's Context with Low-Level Sensors," 6th IEEE International Conference on Electronics, Circuits and Systems, September 5-8, 1999, Cyprus.

[67] Tuulari, Esa, "Context aware hand-held devices," VTT Electronics, 2000. VTT Publications.

[68] H. Harroud, A. Karmouch, R. Impey, "Providing Policy-based Agents Support for Building Mobile Applications," NOTERE June 27 –30 2004, Morocco.

[69] N. Samaan and A. Karmouch, "A Mobility Prediction Scheme based on Dempster-Shafer Theory of Evidence," IEEE Transactions on Mobile Computing, to appear.

[70] Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer, S, "EasyLiving: Technologies for Intelligent Environments," Handheld and Ubiquitous Computing, September 2000.

[71] Krumm, J., et al, "Multi-Camera Multi-Person Tracking for EasyLiving," IEEE Workshop on Visual Surveillance, July 2000.

[72] Shafer, S., Brumitt, B., and Meyers, B, "The EasyLiving Intelligent Environment System," CHI Workshop on Research Directions in Situated Computing, April 2000.

[73] Brumitt, B,. et al, "Let There Be Light: Comparing Interfaces for Homes of the Future," Microsoft Research Technical Report 2000-92.

[74] The CoolTown Website. http://www.cooltown.hp.com/

[75] T. Kindberg, et al, "People, Places, Things, Web Presence for the Real World," The third IEEE Workshop on Mobile Computing Systems and Applications, December 2000.

[76] Caswell, Deborah and Phillippe Debaty, "Creating Web representations for places," In the Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing, Bristol, September 25-27, 2000, pp. 114-126.

[77] Debaty, P; Caswell, D; "Uniform Web presence architecture for people, places, and things," IEEE Personal Communications, Vol. 8, No. 4, Aug. 2001, pp.46 - 51.

[78] Dey, A.K., Futakawa, M., Salber, D., Abowd, G.D., "The Conference Assistant: Combining Context-Awareness with Wearable Computing," Proceedings of the 3rd International Symposium on Wearable Computers, October 20-21, 1999, pp. 21-28.

[79] Salber, D., Dey, A.K., Abowd, G.D, "The Context Toolkit: Aiding the Development of Context-Enabled Applications," Proceedings of the Conference on Human Factors in Computing Systems, May 15-20, 1999, pp. 434-441.

[80] Dey, A.K., "Context-Aware Computing: The CyberDesk Project," Proceedings of AAAI'98 Spring Symposium, Technical Report SS-98-02, March 23-25, 1998, pp. 51-54.

[81] Dey, A.K., Abowd, G.D., Wood, A., "CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services," Proceedings of the International Conference on Intelligent User Interfaces, January 6-9, 1998, pp. 47-54.

[82] Dey, A.K., Abowd, G.D., "CyberDesk: The Use of Perception in Context-Aware Computing," Workshop on Perceptual User Interfaces, October 19-21, 1997, pp. 26-27.

[83] Beresford, A.; et al; "The first summer school on ubiquitous and pervasive computing," Pervasive Computing, IEEE, Vol. 2, No. 1, Jan-Mar 2003, pp. 84 – 88.

[84] Stanford, V, "Pervasive computing goes to work: interfacing to the enterprise," Pervasive Computing, IEEE, Vol. 1, No. 3, July-Sept. 2002, pp.6 – 12.

[85] Steve Vinoski's, "New Features for CORBA 3.0," Communications of the ACM, October, 1998.

[86] Manuel Román, at al, "Gaia: A Middleware Infrastructure to Enable Active Spaces," IEEE Pervasive Computing, Vol. 1, No. 4, Oct-Dec 2002, pp. 74-83.

[87] Manuel Roman and Roy H. Campbell, "Providing Middleware Support for Active Space Applications," In Proceedings of International Middleware Conference, Rio de Janeiro, Brazil, 2003, pp. 433-454.

[88] Manuel Roman, Brian Ziebart, and Roy Campbell, "Dynamic Application Composition: Customizing the Behavior of an Active Space," IEEE International Conference on Pervasive Computing and Communications, Dallas-Fort Worth, Texas, March 2003, pp. 169-176.

[89] Christopher K. Hess, Francisco Ballesteros, Roy Campbell, and M. Dennis Mickunas, "An Adaptable Data Object Service for Pervasive Computing Environments," Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems, San Antonio, Texas, February, 2001.

[90] Christopher K. Hess and Roy H. Campbell, "A Context-Aware Data Management System for Ubiquitous Computing Applications," International Conference of Distributed Computing Systems, Providence, Rhode Island, May 19-22, 2003.

[91] Guanling Chen and David Kotz, "Context-sensitive resource discovery," First IEEE International Conference on Pervasive Computing and Communications, Fort Worth, TX, March 2003, pp. 243-252.

[92] Guanling Chen and David Kotz, "Solar: An open platform for context-aware mobile applications," First International Conference on Pervasive Computing, Zurich, Switzerland, August 2002

[93] Guanling Chen and David Kotz, "Context aggregation and dissemination in ubiquitous computing systems," Fourth IEEE Workshop on Mobile Computing Systems and Applications, June 2002:

[94] Guanling Chen and David Kotz, "Solar: Towards a flexible and scalable data-fusion infrastructure for ubiquitous computing," Workshop on Application Models and Programming Tools for Ubiquitous Computing at Third International Conference on Ubiquitous Computing, Atlanta, GA, October 2001

[95] S. S. Yau, F. Karim, et al, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," IEEE Pervasive Computing, Vol. 1, No. 3, July-September 2002, pp.33-40

[96] S. S. Yau, Y. Wang, F. Karim, "Development of Situation-Aware Application Software for Ubiquitous Computing Environments," Proc. 26th IEEE Int'l Computer Software and Applications Conference (COMPSAC 2002), August 26-29, 2002, pp. 233-238.

[97] S. S. Yau, F. Karim, "Context-Sensitive Distributed Software Development for Ubiquitous Computing Environments," Proc. 25th IEEE Int'l Computer Software and Applications Conference (COMPSAC 2001), October 8-12, 2001, Chicago, USA.

[98] S. S. Yau, F. Karim, "Context-Sensitive Middleware for Real-time Software in Ubiquitous Computing Environments," IEEE Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC 2001), May 2-4, 2001, Magdeburg, Germany, pp. 163-170.

[99] S. S. Yau, F. Karim, "Reconfigurable Context-Sensitive Middleware for ADS Applications in Mobile Ad-Hoc Network Environments," IEEE Int'l Symp. on Autonomous Decentralized Systems (ISADS 2001), March 26-28, 2001, pp. 319-326, Dallas, USA

[100] McGuinness, D.L; "Question answering on the semantic web," Intelligent Systems, IEEE, Volume 19, Issue 1, pp. 82-85, Jan.-Feb. 2004

[101] Michael C. Daconta, Leo J. Obrst, Kevin T. Smith, "The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management," John Wiley & Sons, 2003

[102] John Durkin, "Expert Systems, Design and Development," Macmillan Publishing, 1994.

[103] T. Kanter, "Attaching Context-Aware Services to Moving Locations," IEEE Internet Computing, March-April 2003, Vol. 7, No. 2, pp. 43-51.

[104] T. Kanter, "Going Wireless: Enabling an Adaptive and Extensible Environment," Special issue on Personal Environment Mobility in Multi-Provider and Multi-Segment Networks of the ACM Journal on Mobile Networks and Applications (MONET), Vol.8, No. 1, February 2003, pp.37-50.

[105] T. Kanter, "HotTown, Enabling Context-Aware and Extensible Mobile Interactive Spaces," IEEE Wireless Communications, Vol. 9, No. 5, October 2002, pp. 18-27

[106] T. Kanter, "Extensible Mobile Presence," Proceedings of the 4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN02), September 2002

[107] http://www.w3.org/DesignIssues/Notation3.html

[108] Hong, J.I. and J.A. Landay, "An Infrastructure Approach to Context-Aware Computing," Human-Computer Interaction (HCI) Journal, pp. 2-3, 2001.

[109] Hong, J.I, "The Context Fabric: An Infrastructure for Context-Aware Computing," proceedings of Doctoral Consortium, Human Factors in Computing Systems: CHI 2002. Minneapolis, MN 2002.

[110] Heer, J., Newberger, A., Beckmann, C., and Hong, J.I., "Liquid: Context-Aware Distributed Queries," proceedings of Ubicomp 2003. Seattle, WA 2003

[111] Takayama, L., Leung, L., Jiang, X., and Hong, J.I., "You're Getting Warmer! How Proximity Information Affects Search Behavior in Physical Spaces," Human Factors in Computing Systems: CHI 2003. Fort Lauderdale, FL 2003.

[112] Sim, K.M.; Wang, S.Y.; "Flexible Negotiation Agent With Relaxed Decision Rules," Systems, Man and Cybernetics, Part B, IEEE Transactions on, Vol. 34, No. 3, June 2004, pp.1602-1608.

[113] Perich, F.; Joshi, A.; Finin, T.; Yesha, Y, "On data management in pervasive computing environments," Knowledge and Data Engineering, IEEE Transactions on, Vol. 16, No. 5, May 2004, pp.621 – 633.

[114] Lehti, P.; Fankhauser, P.; "XML data integration with OWL: experiences and challenges," International Symposium on Applications and the Internet, 26-30 Jan. 2004.

[115] Phelps, S.; Tamma, V.; Wooldridge, M.; Dickinson, I.; "Toward open negotiation," Internet Computing, IEEE, Vol. 8, No. 2, March-April 2004, pp.70 – 75.

[116] Bellavista, P.; et al, "Dynamic binding in mobile applications," Internet Computing, IEEE, Vol. 7, No. 2, March-April 2003, pp.34 – 42.

[117] Paolucci, M.; Sycara, K.; "Autonomous Semantic Web services," Internet Computing, IEEE, Vol. 7, No. 5, Sept.-Oct. 2003, pp.34 – 41.

[118] G. J. Klir, et al, "Fuzzy Sets, Uncertainty and Information," Prentice-Hall, NJ, 1988.

[119] Cezary Janikow, "Fuzzy Decision Trees: Issues and Methods," IEEE Transactions on Systems, Man, and Cybernetics, Volume. 28, Issue 1, pp.1-14, 1998

[120] L.A. Zadeh, "Fuzzy Algorithms," Information and Control, Vol. 12, 1968, pp. 94-102.

[121] L.A.Zadeh, "Fuzzy Logic," Computer, Vol. 21, No. 4 ,April 1988, pp. 83 – 93

[122] Vinoski, S., "It's Just a Mapping Problem," IEEE Internet Computing, Vol.7 , No. 3, pp. 88-90, May-June 2003.