An Intelligent Broker Architecture for Context-Aware Systems

by
Harry Chen

A PhD. Dissertation Proposal in Computer Science
at the University of Maryland Baltimore County
January 2003

Abstract

Context-aware computing is an emerging paradigm to free everyday users from manually configuring and instructing computer systems. As the general trend of computing is progressing towards an open and dynamic infrastructure, building context-aware systems can be difficult and costly. In order to build successful context-aware systems, we must develop an architecture to reduce the difficulty and cost of building these systems. This PhD. dissertation proposal describes a research plan to develop a broker-centric agent architecture that is aimed to relieve the burden of capability-limited agents of acquiring and reasoning about contexts, and to protect the privacy of users in a context-aware environment. The implementation of the Context Broker Archiecture will explore Web Ontology Language for modeling contexts and privacy policies, Jess for building a hybrid reasoning mechanism and JADE/FIPA for realizing broker behaviors and agent communications.

Contents

List of Figures

Chapter 1

Introduction

In the last few years, the demand for computing devices and services has rapidly increased. As these devices and services become more complex and sophisticated, everyday users often find themselves spending more time and efforts in configuring and instructing these devices and services. Context-aware computing is an emerging paradigm to overcome these issues. By enabling computer systems to understand their situational contexts, context-aware computing frees users from being the slaves of their computer systems. Thus, it helps the users to do more by doing less [18].

The general trend of computing is progressing towards the vision of Ubiquitous Computing, in which devices are seamlessly integrated into the life of everyday users, and services are readily available to users anywhere they go at any time [50]. In contrast to the traditional computing paradigm, in which the computing environment is often static and well-defined, in Ubiquitous Computing, however, the underlying environment is open and dynamic [44, 50]. Because of these characteristics, building context-aware systems to operate in a Ubiquitious Computing environment will be costly and difficult. In order to develop successful context-aware systems, we must develop an architecture to reduce the difficulty and cost of building these systems.

In this proposal, I will study context-aware computing using intelligent agents [51]. The research aim is to show that the difficulty and the cost of building context-aware systems can be greatly reduced by the use of a broker-centric agent architecture called Context Broker Architecture (COBRA).

1.1 What's Context-Aware Computing?

Framing context-aware computing from the perspective of agents, it is a paradigm for building intelligent agents that can anticipate the needs of users and act on their behalf [13]. In the traditional agent paradigm, the behavior of an agent is fixed for a specific set of environment conditions. In the context-aware computing paradigm, however, the behavior of an agent is designed to adapt to the changes of contexts in the environment.

But what is context? The dictionary definition of context is "the interrelated conditions in which something exists or occurs" [21]. In context-aware computing, context is any information that can be used to characterize the situation of an entity (i.e. a person, computing device, or non-computational physical object) [19]. For the purpose of introducing context-aware computing, let's consider how context plays a role in the following two computing systems:

- Call forwarding [49]: The objective of this system is to automatically forward calls to users and manage incoming calls based on the contexts of users in an office. When the system detects a user is not currently in his office room, it tries to determine his present location in the office and forward calls to a phone that is nearest to that location (location context). In addition, when the system detects the user is meeting with his supervisor in the office, the system automatically forwards calls to the voice mailbox without interrupting the meeting (social context).
- Shopping Assistant [4]: The objective of this system is to enhance the shopping experience of users by exploiting their contexts in a store. In this system, each shopper carries a specialized mobile device. As the shopper wonders around in the store, the mobile device automatically displays the description of the items that the shopper is currently seeing (location context). The device can also recommend sales items that match users' interests without any explicit user instructions (personal context).

1.2 Problems in Context-Aware Computing

Although many context-aware systems have been developed in the past few years [1, 4, 10, 11, 14, 6, 26, 27, 33, 42], the functions of these systems remain to be primitive [12] because building context-aware systems are often difficult and costly. In this section, I will describe three key problems that often arise when building context-aware systems: 1) lack of reusable context-aware mechanisms, 2) limited resources on mobile devices, and 3) privacy issues in accessing user information.

1.2.1 Lack of Reusable Context-Aware Mechanisms

Building context-aware systems from scratch are often costly and difficult. In general every context-aware system requires mechanisms to support context sensing and context reasoning. While context sensing mechanisms deal with the acquisition of information from the physical environment, context reasoning mechanisms deal with the interpretation of the acquired information. Because of lacking reusable context sensing and reasoning mechanisms, many of the existing context-aware systems are difficult and costly to build.

Context sensing, or acquiring information from the physical environment, usually involves hardware sensors [1, 10, 27]. In a dynamic environment, a great amount of hardware sensors are required for agents to acquire a wide divergence of information. Although with the advent of technology the cost of hardware sensors has been decreasing, attaching a large amount of context sensors with individual agents remains to be a costly operation. In addition to the high monetary cost, without reusable mechanisms, directly attach sensors with context-aware agents also makes difficult for building more advanced and complex systems. Although research attempts have been made to develop a reusable sensing mechanism by providing a middle-ware infrastructure between the agents and the sensors (e.g. Context Toolkit [19] and Context Fabric [28]), as context-aware computing progresses towards an open and dynamic environment, new problems arise. In an open and dynamic environment, contexts in question are often distributed in heterogenous sources (e.g. the Web, user profiles, corporate database, personal agents, etc.). Because the middle-ware infrastructure that the agents use to acquire contexts are tightly coupled with the implementation of the agents, modifying the middle-ware infrastructure often requires modifiying the implementation of the agents. When the number of agents in the system grows, any modification to the middle-ware infrastructure could cause the need to modify all agents' implementations, which is a difficult and costly operation.

While it is important to develop reusable mechanisms for context sensing, it is also important to develop reusable mechanisms for context reasoning. Context-aware agents must reason about the information that they have acquired in order to become aware of their contexts. The complexity of any context reasoning mechanism is determined by the complexity of the information that the mechanism attempts to process. Context reasoning mechanisms varies from simple `IF-THEN` programming procedures [43, 19] to complex rule-based programming [14]. With the increasing demand for intelligent systems that can process more advanced contexts (e.g. user intentions, user roles, activities etc.), developing context reasoning mechanisms will become more complex and specialized (e.g. knowledge representation, user modeling, logic programming etc.). Without any reusable mechanisms for context reasoning, building context-aware systems from scratch will be difficult and costly.

1.2.2 Limited Resources in Mobile Devices

With the rapid technology advance in mobile computing, mobile devices are gradually becoming an intricate part of the future computing infrastructure. While mobile devices have been the core of many of the existing context-aware systems, these devices often hinder the functionality of their respective systems, making difficult to build more advanced and complex system behaviors. This is mainly due to limited computational resources on mobile devices, such as battery power, information storage, computing power, and communication.

Battery Power Constraint

Context-aware agents often rely on sensors to acquire contexts. However, most of the mobile devices are not designed to support sophisticated sensing hardware due to their battery power constraint. Although researchers have successfully integrated sensors on some advanced mobile devices such as Pocket PC [27], the type of contexts that can acquired using these sensors remain to be basic and limited. It might be argued that as the mobile computing technology advances, more powerful devices such laptops and wearable computers will have less restricted battery power. Thus, it will be possible for these devices to be equipped with sophisticated sensors. However, there is one problem with this argument. The use of mobile devices is meant for convenience. When mobile devices are attached with multiple external sensors that are awkward and clumsy, they will loss their basic function as convenient computing devices.

Information Storage Constraint

Context-aware agents often need to store acquired contextual knowledge in order to reduce the cost of performing repetitive context acquisition and to exploit the historical information of contexts from the past.

On mobile devices, the available disk space for storing information are often limited. Agents usually do not have the privilege to continuously store all knowledge that are acquired as they provide round the clock services for users. In order to work with limited storage space, agents will be faced with the challenge to be selective about what kind of knowledge should be stored and what kind of knowledge should deleted. Furthermore, because contextual knowledge acquired from physical environment are potentially inaccurate, agents will also be challenged to maintain knowledge consistency and resolve information ambiguity.

Computing Power Constraint

Knowledge management and context reasoning are both computation intensive processes. They often require substantial computing power (i.e. CPU power and memory). Because mobile devices are often lack of CPU power and memory, context-aware agents on mobile devices can only process primitive contexts [12].

One solution to this problem is to off-loaded the computation intensive processes to some resource rich stationary computers [43, 19]. However, this raises the issues in the communication constraint of mobile devices.

Communication Constraint

In a dynamic environment, the communication between agents on mobile devices and the nearby context sources often cannot be pre-defined. This problem is twofold. First, agents may be lack of sufficient knowledge to communicate with context sources in the environment (e.g. knowing which sensor can provide what information and how to communicate). Second, context sources may dynamically join and leave the environment without notifying the agents. For these two reasons, it will be difficult for context-aware agents on a mobile device to effectively acquire contexts.

1.2.3 Privacy Issues in Accessing User Information

People have always been concerned about privacy issues in computing systems. Especially, people are worried about how computer systems use and share their personal information. In a context-aware environment, agents often collect and share user information. This raises great concern for user privacy.

Privacy is intrinsically bound up with control - who controls what information as well as the applications (agents) that construct and disseminate that information [2]. In the existing context-aware systems [19, 30, 33, 4], users often do not have control over how personal information are to be acquired. As context sensors are built to be hidden in the physical space, personal information are collected without the explicit consent of the users. This is a serious privacy problem.

Privacy issues also arise when sharing user information. Let's consider the following example [2]: a user walks into a conference room. The air-conditioning agent wishes to adjust the temperature appropriately, and thus asks the room agent to acquire the identify of the user. The room agent asks the user for his identity. At the same time, a file caching agent wishes to provide some file caching services for the user, and thus it asks the room agent for the user information. The problem here is that when the user has provided the room agent of his identity for the purpose of adjusting room temperature, there is no way for the user to know further uses of his identity information provided to the file caching agent. In another word, the *downstream consequences* for providing private information are unknown or unspecified [2].

1.3 Proposed Solution: Context Broker Architecture

In order to reduce the cost and difficulty of building context-aware systems, I propose to develop a broker-centric agent architecture (i.e. COBRA) to provide runtime supports for context-aware systems in an Intelligent Meeting Room environment[1]. The design of the COBRA architecture is aimed with the following two objectives:

1. Enabling distributed context-aware agents (possibly running on resource-limited mobile devices) to contribute to and access a shared model of the context

[1]Intelligent Meeting Room is a type of Ubiquitious Computing system which emphasis on the any time, anywhere computing paradigm in office meeting rooms. Intelligent Meeting Room research has close relationship with Intelligent Room [10] and Smart Space [24] research.

2. Allowing users to control the access of their personal information in a context-aware environment

The core of the COBRA architecture is a broker entity called *Domain Context Broker* (or broker for short). Broker is an autonomous agent that manages and controls the context model of a specific domain (e.g. an office, a conference room, or a department). In a domain, the broker has the following responsibilities:

1. Maintaining the context model of the domain, which includes domain contexts from the past and at the present

2. Resolving inconsistancies and ambiguities of the domain contexts through information fusion

3. Establishing privacy policies with users before sharing their personal information

4. Providing knowledge sharing service for context-aware agents through agent communications

1.3.1 Use Case

As the COBRA architecture finds its application in an Intelligent Meeting Room environment, let's consider a use case of the architecture.

After purchasing the latest version of COBRA implementation, the system administrator installs Domain Context Broker in every rooms of the company (i.e. conference rooms, office rooms, cafeteria etc.). Each broker is loaded with specific domain knowledge of the company (e.g. maps of the building, organizational charts etc.) and is connected to various sensors (e.g. badge sensors, video cameras, corporate messaging services etc.) for acquiring contexts from the physical environment.

One day afternoon, a scheduled meeting is about to begin. As Alice and Bob enters the conference room, the broker in the room detects the presence of the personal agents that run on the devices that Alice and Bob are carrying (e.g. cell phones, RDIF badges, PDA, laptop etc.). Subsequently the broker concludes Alice and Bob are currently present in the room.

Before the broker shares these information with any agents in the system, it negotiates privacy policies with Alice and Bob for sharing personal contexts. Alice's policy specifies that her personal contexts can be shared with her personal agents and any agents that provide meeting related services, as long as she is attending the meeting. Bob's policy, on the other hand, specifies that none of his personal contexts should be disclosed to any agents excepts the meeting minutes-recording agent and his personal agents (running on remote Web sites and local mobile devices). Knowning the policies of these two users, the broker informs their personal agents of their location contexts.

Meanwhile, as other participants arrive at the meeting, from contextual information acquired from various sources (e.g. the Web, the corporate database, meeting schedule, user profiles etc.), the broker constructs the context model of the meeting domain. This model includes contextual information about people who attend or organize the meeting, the location context of the participants, the activities that are expected to happen during the meeting (e.g. presentation, discussion, conference calls, demo session etc.) and the roles of different participants during the meeting (e.g. speaker, audience, organizer, visitor etc.). Knowning that Alice has a role of being a speaker and will give a presentation at the begining of the meeting, the broker informs all agents in the room that wishes to assist Alice for presentation.

Among all the agents that are being informed, the projector agent wishes to help Alice to setup the presentation. In order to upload the correct slides to the projector, the agent asks the broker for information. Not knowing the answer, the broker asks Alice's laptop agent for slides that Alice is expected to give a presentation on. The laptop agents replies with appropriate information. After updating the context model of the domain, the broker immediately relays that information to the projector agent. Upon receiving information about the presentation, the projector agent automatically setup the slides and dims the lights before the presentation starts.

After the presentation, participants decide to have a short break. Leaving his personal device behind, Bob goes to the cafeteria for a cup of coffee. As he enters the cafeteria, the local broker detects his presence in the cafteria and informs his personal agent on the Web of his current location context. At this moment, without being informed that Bob has ever left the conference room, Bob's personal agent on the Web detects knowledge inconsistency of Bob's location context. Immediately, the agent reports this critical problem to both brokers of the conference room and the cafeteria. After sharing evidences with each other, the brokers resolve knowledge inconsistency in regard to Bob's location context and informs his personal agent of the correct information.

1.3.2 Research Contribution

In the user case described in the previous section, if the context-aware agents (i.e. the personal agents and the projector agent) operate without the support of the COBRA architecture, then it is clear that each agent must be developed with specialized context-aware mechanisms. Because the COBRA architecture will provide agents with a suite of essential context-aware mechanisms, the developers of the agents can concentrate on the functions of more advanced context-aware behaviors rather than specific context-aware mechanisms.

The contribution of this dissertation will be to show that the difficulty and cost of building context-aware systems can be greatly reduced by the use of the COBRA architecture. This broker-centric agent architecture will demonstrate the following key features:

- **Context aquisition in Semantic Web**: in a dynamic environment, physical sensors can only offer agents with limited access to contexts. To widen the types of contexts that can be accessed by the agents, the CO-

BRA architecture will exploit the emerging Semantic Web infrastructure and technolgy. In specific, the architecture will demonstrate how Semantic Web langauges such as OWL (Web Ontology Language) and RDF/RDFS can be use to model and reason about contexts in an Intelligent Meeting Room environment.

- **Hybrid context reasoning mechanism**: maintaining consistent and coherent model of context in a dynamic environment requires advanced context reasoning mechanisms. In the previously developed context-aware systems, context reasoning has always been built on *ad hoc* procedures with deductive reasoning in core. To improve upon the existing approach, the COBRA architecture will integrate the context reasoning with other means of machine reasoning (e.g. heuristic reasoning, fuzz logic, statistic analysis, user modeling etc.) to build a hybrid context reasoning mechanism.
- **Knowledge sharing through context models**: when acting in isolation, agents have limited access to context. To overcome this problem, the COBRA architecture will extend the standard FIPA architecture [22] to allow distributed agents to contribute to and access a shared model of context. Through knowledge sharing, context-aware agents can effective detect and resolve inconsistent contextual knowledge.
- **Policy-driven privacy protection**: in a context-aware environment, using or sharing users' personal contexts without the consent of the users violate user privacy. In order to allow users to control how their personal contexts can be used and shared, the COBRA architecture will define a policy-driven mechanism to protects the privacy of users. This mechanism allows users to negotiate with the context-aware systems to form policy rules that defines how personal contextual information can be used and shared.

In addition to the design of the COBRA architecture, along with required ontologies, APIs and protocols, I will also demonstrate the feasibility of this broker-centric approach by developing a prototype system. This system will be evaluated thru a series of experiments in an Intelligent Meeting Room scenario (see Sec. 4.2).

1.4 Proposal Outline

- In Ch. 2, I will discuss the notion of context and context-aware computing research. In addition, I will examin the future research direction of context-aware computing in the light of Semantic Web.
- In Ch. 3, I will describe a detailed design of the Context Broker Architecture. The discussion will cover how COBRA organizes contexts, how broker maintains a model of context, negotiate privacy policy with users, and sharing contextual knowledge with agents.

- In Ch. 4, I will summarize this document and present the preliminary research approach. A feasibility study will be described for the purpose of evaluating the COBRA architecture. At the end, the milestones of the future research will be presented.

Chapter 2

Background

2.1 Reasons to Study Context

We humans often exploit context when we communicate and take actions. Because we are context-aware beings, we are able to effectively convey ideas without needing to explicitly state background information during conversations. For example, when two people walk into the same room, one person tells the other person, "Close the door, please". Because they share a common context, there is no need for the first person to explicitly point out the door he intends to close. Context can also guide us to adapt behavior in changing environment conditions. For example, in a movie theater, when a movie starts to play, we usually try to avoid loud conversations; when driving on a highway, as traffic becomes heavier, we usually slow down in order to avoid hitting other cars.

Because the notion of context effects the intelligent behavior of the humans, it is generally agreed that context-awareness should be simulated in computing systems to strengthen their capability to communicate, to behavior and to process information. Context has been studied in many different fields of computer science, including Artificial Intelligent [36, 48], Information Retrieval [11], Nomadic Computing [30], Sensor Networks [46, 40] and Ubiquitous Computing [42, 43, 3, 13]. Although different fields of research have studied contexts with distinctive aims, they all recognize that context-awareness is the vehicle to the realization of intelligent computing systems.

2.2 Definitions of Context

In order to engineer computing systems that are context-aware, it is important to understand what constitutes context from an engineering perspective. In another word, how should the notion of context be defined when developing a context-aware system?

In Ch. 1 we have seen the general definition of context in a dictionary is "the interrelated condition in which something exists or occurs". Some researchers argue that the dictionary definition does not offer much understanding about how context is related to computing environment [12]. Thus, a more precise definition of context must be developed for building context-aware systems.

Schilit *et al.* characterize context as a collection of information that describe the users in a context-aware system. Schilit describes context as the following [43]:

Definition 1 *In a mobile distributed computing system, contexts are the location of the user, the identity of people and physical objects that are nearby the user, and the states of devices that the user interact with.*

While this definition characterizes the types of contexts that Schilit *et al.* have developed [43], but it does not cover any other types of contexts that are useful for building context-aware systems (e.g. the intentions of the users, the activities that the users are participanting etc.). Dey argue that a definition of context should not just be a list of information that describes users or the system because "*context is all about the whole situation that is relevant to an application and its set of users*" [19]. Because the situation relevant to an application and its users consists of potentially infinite number of information [12], it is not suitable for the definition to be just based on the enumeration of these information. Dey gives a different definition of context [19]:

Definition 2 *Context is any information that can be used to characterize the situation of an entity. An entity is a person, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.*

It is clear that Def. 2 is more general in comparing to Def. 1. While it does not bind to a specific list of information, Def. 2 extends the notion of context to include any information that describes physical objects and computing applications in addition to users. Besides Schilit's and Dey's definitions of contexts, few other researchers also have attempted to describe context from a computing persepective [46, 12]. All of which are closely in spirit of the formers.

Although a number of definitions of context have been offered, but none so far has been adopted as the standard definition of context for context-aware computing. In my opinion it is likely that multiple definitions of contexts will co-exists for various reasonings (e.g. political or technical). Subscribing to the thinking of McCarthy and Buvac [36], I believe that it is acceptable for various definitions of context to co-exist if they are all useful for building effective computing systems. However, it is important to recognize that when adopting a general definition of context for the development purpose, it is still necessary for developers to further specify or enumerate a list of contextual information that will give design specification of the system. In other word, I suggest a combination of Schilit's and Dey's approaches to define context.

In this document, I will introduce a modified version of Def. 2 as the definition of context for developing the COBRA architecture. This definition (Def. 3) aims to describe the following four piecie of contexts in an Intelligent Meeting Room environment:

1. Who are the people in the environment?
2. Where are these people (in the past, now, and in the future)?
3. What devices are around or carried by these people?
4. Why are these people and devices in the environment?

Definition 3 *Context is any information that can be used to characterize the situation of an entity. An entity is a person, a computing device or a non-computing physical object that is considered relevant to the interaction between a user and an application. In an Intelligent Meeting Room environment, contexts are*

1. *the identity of people and devices,*
2. *the location of the people and devices,*
3. *the activities that the people are participanting in, and*
4. *the roles and intentions of people when participating in the activities.*

2.3 Aspects of Context-Aware Computing

2.3.1 Enhancing User Interfaces

One aspect of context-aware computing is aimed to enhance the user interface of mobile devices. Mobile devices become increasing important in our everyday life. Although these devices give us the freedom to exploit computing services without constraining us to sit in front of the desktop computers, they also raise new challenge in designing user interfaces. At any time, the users can be simultaneously engaged in real-world activities while interacting with the mobile devices, for example walking down a busy street, talking to other people, and driving a car. Because typical sessions with these devices may last seconds or minutes rather hours, the design of user interfaces must be minimally disruptive and minimally demanding of cognitive and visual attention [27].

For today's mobile devices, the size of a device often constrains the design of its user interfaces. These devices often have tiny buttons, small display screens. Users often feel awkward to interact with the device through these miniature interfaces. For example, finding a button or activating a control on the screen can require significant visual attention [27].

To overcome this problem, Hinckley *el at.* have developd a context-aware approach to replace some traditional user interface on a palm-size PC (i.e. Cassiopeia E-105) [27]. In their approach, the functions of the device can be controlled by the way how a user holds the device, the position of the device and the orientation of the device. For example, when the on-board sensors detect the user is holding the device like a cell phone or microphone and speaking into the device, then the device will automatically active the voice memo application. When the sensors detect the user is holding the device in a landscape orientation as oppose to portrait, the device will automatically reformats the display to suit the current viewing orientation.

Schilit *et al.* [44, 43], Rekimoto [41], Harrison *et al.* [26] also have develop similar approaches to enhance the user interfaces on mobile devices. Schmidt *et al.* [45] describe a cell phone that combines tilt, light, head, and other sensors to sense the context of the device such as sitting on a table, in a briefcase, or being used outdoors. These states are exploited to automatically adjust the tone and the volume of the ring. Rekimoto uses the tilting position of the device for guiding menu selection. Harrison *et al.* exploits the tilting position to help users to scroll through the information on the device display.

2.3.2 Guiding the Adaptation of System Behavior

In an open and dynamic environment, conditions are constantly changing. Context is useful for guiding systems to adapt their behavior. The key characteristic of the future computing world is dynamic. Because the interactions between users and computers will no longer be restricted in a fixed environment, user devices and computing systems will need to be able to dynamic adapt their behaviors. For example, when the network bandwidth changes, the video streaming application on a wireless PDA must be able to dynamically adjust the streaming quality without interrupting the viewer's attention [44]; in a public social environment, when not all people are trusted users, a document sharing service must be able to dynamically adjust the policy for sharing sensitive documents based on the role of the users [14].

Network properties are commonly used contexts for guiding the behavior change of mobile applications. These properties include network bandwidth, error rate, connection setup time, usage costs, security requirements, contention, disconnection rate, and round-trip delay [44]. Among these properties, network bandwidth is the mostly used. For example, the context of network bandwidth can guide the video player application on a mobile device to automatically adjusts the quality of video to play without interpreting the streaming of the video, and it can also guide a web browser to decide the right image files to request from the web server without increasing the downloading time [37].

2.3.3 Enabling Smart Space Applications

Context also plays important role in developing intelligent applications in Smart Space research. The research is developed on the belief that in order for ubiquitous computing to be successful, we must develop computer systems that draw computing into the natural world of the humans, as oppose to drawing humans into the complex world of computers [25]. In the context of Smart Space, the notion of context provide a means for computer systems to automatically reason about the situation of the human users. Thus, it allows systems to anticipate the needs of users and act on their behalf.

User location is a commonly used context in Smart Space research. In Cyberguide [19], the location of a tourist is exploited by the system to provide direction services and interactive map services. In the call forwarding system described in Ch. 1, the system exploits the location of employees when routing incoming calls in an office. Bennett *et al.* [6] describes a teleporting system which exploits user locations to enable the user interface of applications to dynamically migrate from resource-poor mobile devices to nearby resource-rich devices. In Intelligent Room project [15], the location of a user in a room is used in conjunction with other context to determine the user's intention.

User identity is another commonly used context. The identity of the user is often mapped from some kind of physical objects that the user carries (e.g. RFID badge, Active Badge, and PDA) [43, 19, 34]. From user identity information, systems can provide customized services without requiring explicit inputs from the users, and thus minimizes the efforts on the side of the users to manually configure and instruct the systems. Asthana *et al.* [4] describes a shopping assistant system that uses user identity information to distinguish shoppers from *regular customers* who are anonymous and *store customers* whose identities are known by the system.

In a few Smart Space systems, the intention of a user have been explored as one type of user context. Unlike user location and identity contexts, user intention cannot be easily acquired from direct sensing. Customized reasoning mechanism is required to infer user intention from conditional information gathered from the environment. In the Intelligent Room project, user intention is inferred from the behavior model that is associated with the user [15]. For example, a user can be walking, sitting, standing or pointing in a meeting. Depends on the noise level and people motions, the system can conclusion about what the user is current doing.

2.4 How Do Applications Acquire Contexts?

Context is any information that can be used to characterize the situation of a person, a device or a non-computing physical object. To enable context-awareness, system developers must provide agents with the access to context. The process of acquiring context from the physical environment is called context acquisition. This section reviews three different approaches that enable context-aware agents to acquire contexts:

1. Contexts are acquired by directly accessing low-level context sensors.

2. Contexts are acquired from some kind of middle-ware infrastructures that in turn interact with low-level context sensors.

3. Contexts are acquired from servers that maintain situational knowledge about the environment.

2.4.1 Direct Access to Hardware Sensors

To access contexts in the real world requires sensors. Thanks to the advance of sensing technology, a diverse type of contexts can be made available to context-aware agents. Many mobile devices and system applications have exploited this opportunity to bring about context-awareness.

In the design of a context-aware user interface for palm-size PC [27], the device acquires contexts by directly accessing three different type on-board sensors: proximity range sensors, touch sensors, and tilt sensors. From the proximity range sensors, the device can determine a proximate distance between a physical object in the range and the device. From the touch sensors, the device can determine whether a user is holding the device and how long it has been held. From the tilt sensors, the device can determine the tilt angles of the device (i.e. left/right and back/forward), the display orientation of the device, whether the device is being shook etc.

In the design of the Forget-me-not devices, the device directly access Active Badge sensors to determine the location of the users and people whom the users have encountered [33]. Using Active Badges, in the call forwarding system described by Want *et al.* [49] and in the teleporting system described by Bennett *et al.* [6], context-aware agents also directly access the sensors for the up-to-date location information of the users.

Langheinrich *el at.* describes a RFID Chef application that recommends cooking recipes for available foods on the kitchen table [34]. In this system, the agent directly access RFID sensors to identify the type of food that is available.

It is clear that all systems described in the above acquire context from a diverse type of sensors. However, they all have one theme in common: from the raw data provided by the low-level sensors, individual agents apply their own interpretation of contexts. This approach has some potential problems.

First, in order to extend an existing agent to consider additional contexts, the agent must be modified and the context acquisition procedures must be rewritten. Second, because the context acquisition procedures are tightly coupled with the application implementations, such design discourages code reuse.

2.4.2 Facilitated by a Middle-ware Infrastructure

To overcome the extensibility and reusability problem impose by tight coupling sensor accessing and application implementation, middle-ware approaches have been proposed to facilitate context acquisition. The underlying principle in the middle-ware approach is to separate the low-level sensing details from the high-level agent implementations.

In Odyssey [37], agents on mobile devices rely on a middle-ware infrastructure called Odyssey client to provide context about the communication network. An event notification approach is used in Odyssey to notify agents about the context changes. To acquire context, agents running on the device first specifies a *window of tolerance* (the context that the agents are interested), and then registers a event notification with the Odyssey client along with the window of tolerance. When the client discovers that the availability of a resource has strayed outside a registered window of tolerance, it generates a upcall (notification) to the corresponding agents. Using this architecture, Noble *et al.* have develop three context-aware agents (i.e. a video player, a web browser, and a speech recognizer) that can adapt their behaviors in changing of network bandwidth.

Similar to the design of Odyssey which shields high-level implementations from low-level context sensing details, Context Toolkit [19] is another middleware infrastructure that facilitates context acquisition. Inspired by the design of graphical user interface (GUI) toolkit which shields interface interaction details from the applications, the Context Toolkit is designed to shield context sensing details from the agents. The design of the toolkit builds on the *widget* concept. The toolkit defines a collection of widgets called *context widgets* to deal with low-level sensing details. For example, the `IdentityPresence` widget provides callbacks for agents to notified about the departure and arrival information of people. This widget in turn accesses iButton and TIRIS RF tags for context sensing. Using widgets provided by the Context Toolkit, a number of context-aware agents have been developed, for example, the In/Out Board which tracks the presence of people in an office, and the DUMMBO Meeting Board which is an instrumented digitizing whiteboard that supports the capture and the access of meeting minutes.

Both Odyssey and Context Toolkit have taken a middle-ware approach to facilitate context acquisition. Because the low-level sensing of contexts are shielded from the high-level applications, both architectures make easy for developers to introduce new contexts in an existing applications. Thus, they promote the reuse of context acquisition mechanisms.

However, in a dynamic environment, these middle-ware approaches face knowledge consistency problems when supporting a large-scale of system. In a large-scale system, there are a vast number of agents. With any middle-ware approach, individual agent can only access context through their own context acquisition components. For example, in Odyssey, agents on a mobile device can only access the Odyssey client that are built-in to the device operating system; in Context Toolkit, agents can only access designated context widgets in the local environment. Because context sensed by the low-level sensors can be noisy and possibly inaccurate [20], in the absence of a shared memory, different agents can potentially possess inconsistent or conflicting knowledge about the same context.

Moreover, complex middle-ware infrastructures often cannot be built into devices with limited computing resource. Sensing context and interpreting context are often computation intensive operation (e.g. communicating with hardware sensors, computing error correction for noisy sensed data). Although the programming codes for context sensing and interpretation are abstracted for the agents in both Odyssey and Context Toolkit, but all computation of contexts are processed on the same computing platform as the agents. As the complexity of context sensing increases (i.e. more sensors and more sophisticated context interpretation), resource-poor devices cannot possibly accommodate the resource demands of the middle-ware infrastructures.

2.4.3 Acquiring Contexts from a Context Server

In order to support a large number of context-aware agents (potentially with limited resources) in a dynamic environment, an alternative approach is to enable agents to access contexts from a server entity (or context server). The idea is to relieve the burden of sensing context and computing context completely from the agents by shifting these tasks into a resource-rich server. Running on a resource-rich computing platform, context server is a server process that maintains contextual knowledge on the behalf of the agents.

Enabling context reasoning is the main advantage of the context server in comparing to the previous two. Context reasoning is a process in which contextual knowledge acquired from low-level sensors are aggregated to deduce additional knowledge that otherwise cannot be directly sensed using hardware [14]. For example, the knowledge about the roles of different people in meeting cannot be directly acquired using conventional sensors like RFID badges. Such knowledge can only be deduced through a systematic reasoning process. For example, when determining the context of a meeting, the knowledge about the presence of people often need to be reasoned in conjunction with the background knowledge of an on-going meeting [44, 19, 14].

Given the vast number of things in the real world that can have contexts, the design of context server often divides the physical world into sets of micro-world. In the design of the Me-Centric Domain Server (or domain server for short), these micro-worlds are called domains [39]. The notion of domains provides domain server a means to effectively reason about contexts. For the domain that is managed by the domain server, customized reasoning rules (i.e. RDF/RDFS axiom rules) are defined to deduce contextual knowledge (i.e. the location context of the people in room).

To support context-aware applications running on resource-poor devices, applications can acquire context from the server through high-level communications. In the Me-Centric Domain Server, communications are defined in terms of speech act primitives. Using these defined primitives, applications can query the domain server for desired contextual knowledge. In addition, the domain server also allows internal contextual knowledge to be modified by applications through communications (i.e. add/remove/update facts).

There are similarities and differences between the context server approach and the other two approaches described in this Sec. 2.4.1 and Sec. 2.4.2. In particular, both the context server approach and the middle-ware approach can overcome the tight-coupling problem of the direct sensor access approach. Like the middle-ware approach, the context server approach also can separate the low-level sensing details from the high-level implementations. On the other hand, the context server approach can overcome the limitation of the middle-ware approach in supporting a largr number of agents that are potentially with limited resources. However, there are general concern for the centralized design of the context server (i.e. single point of failure). Kumar *el at.* has addressed this problem by developing a team of servers with pre-defined cooperation protocols (i.e. Joint Intention) to achieve fault-tolerance [32].

2.5 Context-Aware Computing Meets Semantic Web

In the past few years, the Web is gradually becoming a de facto standard for people to share information, for business to provide services and for users to communicate with their devices. As this trend continues, the Web will create a number of new opportunities for context-aware computing.

First, the emerging Semantic Web standards will enhance the communication and information sharing between context-aware applications. Second, the Web will become a major source of information for determining the context of users and their associated daily activities.

Semantic Web is a vision of the next generation Web infrastructure. In this vision, both web pages and web services will all be annotated with semantic mark-ups. These mark-ups provide a means for computer systems to "understand" the content of the web pages and the operational details of the web services [8]. The Semantic Web standards emphasis on ontology representation as a means for describing web contents and services. An important part of the standards is the semantic mark-up languages for representing web ontologies. Built on XML encoding, Web ontology languages such as RDF/RDFS [35, 9], DAML+OIL [8] and OWL [17] are some of the popular mark-up languages for constructing web ontologies.

As Web ontology languages emerge, they will play important roles in the communication and information sharing between context-aware agents. Effective communication between independently developed agents requires sharing of common ontology. In the existing context-aware systems, domain ontologies are often built into the specific implementation of the systems. Thus, it is extremely difficult for independently developed systems to interoperate, communication and share information. Web ontology languages could a solution to this problem. Through explicit modeling of domain ontology, context knowledge are represented and reasoned in separate of the system implementations. A declarative representation of context allows independently developed agents to share and exchange information and thus achieves interoperability.

Another important role of Semantic Web is context acquisition. In today's Internet, the Web is one of the most effective medium for people to acquire information, whether it is information about a person, a project, a service or an event. As the Semantic Web technology matures, the future Web will be populated with vast amount of information that will be invaluable for determining contexts. For example, let's consider a person who lives in a Cooltown environment [30]. When this person enters a room, his location information (his presence in the room) is automatically updated in his Web Presence Manager (his personal web site). Immediately, context-aware agents in both remote and local environment can acquire the presence information about the person through the Web without needing any sensing or reasoning. From this simple scenario, it is clear that the future Web technology will be invaluable to context-aware computing. Not only because it enables independently developed agents to interoperate but also opens a new frontier for agents to acquire contexts.

Chapter 3

Context Broker Architecture

Building context-aware agents can be difficult and costly without the support of any computing infrastructure. In order to reduce the cost and difficultites of building context-aware agents, we must develop an architecture to enable distributed agents to contribute to and access a shared model of contexts and to allow users to control the access of their personal information in a context-aware environment.

In this chapter, I will describe the design of the COBRA architecture. Extending the FIPA agent platform, this architecture is aimed to relieve the burden of context-aware agents of acquiring and reasoning about context and to provide privacy protections for users in a context-aware environment.

3.1 Objectives

The overall design objective of the COBRA architecture is to reduce the cost and difficulty of building context-aware agents. The design of this architecture will demonstrate the following four key features:

1. **Acquiring context from heterogeneous sources**: in order to support the functions of advanced context-aware behaviors, it is necessary for agents to consider information from a wide range of sources (e.g. the Web, user profiles, behavior patterns etc.), not just from sensors that are embedded in the local environment. To help capability-limited agents, the COBRA architecture will acquire information from heterogeneous sources and reason about contexts on the behalf of these agents.

2. **Maintaining consistent contextual knowledge**: in a dynamic environment, capability-limited agents cannot maintain consistent knowledge without the support of computing infrastructure. In order to prevent these

agents from making inaccurate decisions due to inconsistent knowledge, the COBRA architecture will detect and resolve any contextual knowledge that may be inconsistent or ambigious on the behalf of the agents.

3. **Enabling knowledge sharing among agents**: when acting in isolation, the knowledge of a single agent is limited. In a dynamic environment, it is more cost-effective for agents to share their knowledge. Knowledge sharing requires cooperation. Because independently developed agents do not have pre-defined cooperation to share knowledge, the COBRA architecture will enable these agents to share knowledge through high-level agent communication.

4. **Protecting the privacy of users**: context-aware agents will rely on the support of COBRA to acquire contexts. To protect user information from being misused by context-aware agents, the COBRA architecture will provide mechanisms for users to control how their personal information are to be shared and used by agents in the environment.

3.2 Modeling the Physical World in Domains

Because the number of contexts that can be described are potentially infinite in the real world, it raises the question of *how to effectively model and structure contexts?*

Taking the traditional "divide and conquer" approach, the world can be divided into a collection of micro-worlds that allows contexts to effectively modelled and structured. In the COBRA architecture, these micro-worlds are called *domains*. Each domain is a knowledge model (or context model) about a partial world in which people, devices and non-computing physical objects interact.

The context model of domain consists of the contextual information that describes 1) the identity of people and devices in the domain, 2) the location of these people and devices, 3) the activities that these people are participating in, and 4) the roles of and intentions of these people when participating in the activities. The description of these contexts follows the definition of context given in Def. 3 in Ch. 2.

The term domain defined here is similar to the definition of domain defined in the Me-Centric Domain Server [39]. However, unlike the previous definition, the COBRA architecture makes no distinction between the types domains that make up the real world.

3.3 An Example System of the Context Broker Architecture

In the COBRA architecture, a domain represents a part of the real world on which a context-aware system can be built. A context-aware system may consists of multiple domains. In each domain, there is an autonomous agent called

Domain Context Broker. The role of the broker is to enable context-aware agents to share a common model of context and to protect the privacy of users in the environment. The details of a broker is described in Sec. 3.4.

Let's consider an example of a COBRA system of a Computer Science department (see Fig. 3.1). This system consists of six different domains. The context of each domain is managed by a broker. To make their presence to be known by other agents, the brokers register themselves with the FIPA Agent Directory Services.

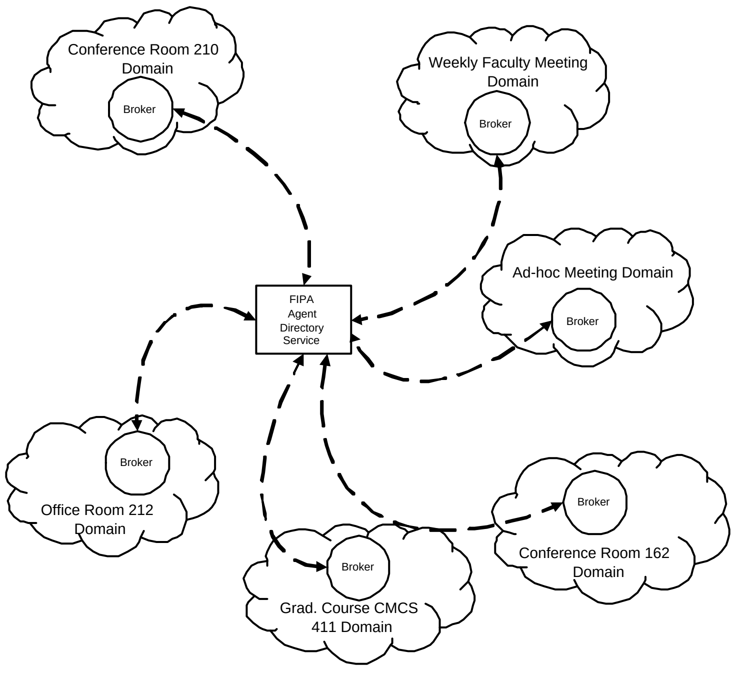
Conference Room 210 Domain
Broker
Weekly Faculty Meeting Domain
Broker
Ad-hoc Meeting Domain
Broker
FIPA Agent Directory Service
Broker
Office Room 212 Domain
Broker
Conference Room 162 Domain
Broker
Grad. Course CMCS 411 Domain

Figure 3.1: An example of COBRA system

It is important to note that in a COBRA system, there is no fixed rules for deciding the organization of domains. System developers are responsible for making such decisions. For example, in Fig. 3.1 there are two distinctive meeting domains (i.e. Weekly Faculty Meeting Domain and Ad-hoc Meeting Domain). However, it is completely legitimate for a system developer to decide combining these two domains into one single domain (e.g. Meeting Domain).

Someone may argue that this definition of domain seems too loose and general. It may create potential arguments for deciding whether some part of the world should be an independent domain or should be a part of some larger domain. However, this problem can be solved through standardization. As the development of Ubiquitious Computing matures, it is likely that there will be standard ontologies for describing intelligent systems and the physical environment. With these standard ontologies, the arguments for whether some part of the world should be an independent domain or not will simply be a question of which standard ontology should one adopt.

3.4 The Design of Domain Context Broker

Domain Context Broker is the core of the COBRA architecture. Brokers are always pre-defined for each domain in the system. In specific, the profile of these brokers (i.e. how can an agent communicate with the broker, which domain does a broker belong to etc.) are pre-defined knowledge of agents in the system. A broker has the following four responsibilities:

1. **Context Acquisition and Fusion**: the broker will fuse contexts by acquiring information from heterogeneous sources. During the process of context fusion, the broker will continuously monitor information changes in different sources and pro-actively acquire missing information as sources becomes available.

2. **Knowledge Maintenance**: the broker will maintain the context model of the domain. During this process, the broker will continuously revise its contextual knowledge for the purpose of ensuring a consistent and coherent knowledge base. This process will be triggered either when some inconsistency knowledge are detected or when some users choose to remove their personal information from the knowledge base due to privacy concerns.

3. **Knowledge Sharing**: the broker will enable independently developed agents to share contextual knowledge through agent communications. The broker will allow agents to query contextual knowledge of the domain and to register event notifications for context changes. Through knowledge sharing, capability-limited agents can contribute to a shared model of contexts and can access context that are otherwise inaccessible if they act in isolation.

4. **User Privacy Protection**: the broker will negotiate privacy policy with individual users before sharing any contextual knowledge with other agents. The privacy policy of a user will define what personal information the user is willing to share with agents in the system, under what circumstances specific information can or cannot be shared etc. When sharing information with untrusted agents, the broker will attempt to protect the privacy of the users by assigning anonymous identities to replace the true identities of the users.

The above four responsibilities of the broker define the functional requirements of the broker. To meet these requirements, Fig. 3.2 shows a conceptual design of a Domain Context Broker. The following subsections will overview the design of each component in the figure.

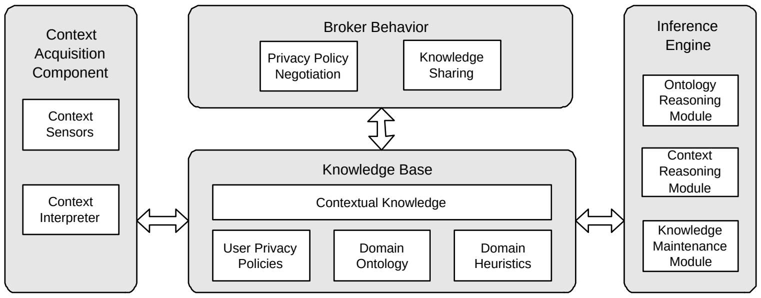

Figure 3.2: The conceptual design of Domain Context Broker

3.4.1 Knowledge Base

The Knowledge Base of a broker is a data repository for both context model of the domain and system knowledge of the domain. The make-up of context model is simply contexts defined in Def. 3 (see Ch. 2). The make-up of system knowledge consists of **Domain Ontology**, **Domain Heuristics** and **User Privacy Policies**.

Domain Ontology

Domain ontology is a set of well-defined vocabularies, concepts, functions and relationships that enable the broker to reason and to behave. In specific, this ontology is divided into four distinctive categories:

1. Context Model: this model covers the fundamental concepts for representing contexts. It allows the broker to reason about contexts and provides a means for the broker to share knowledge with agents in the system.

2. Information Source Model: this model defines the information sources from which the broker can acquire contexts. It provides a specification for the broker to communicate with sensors, users devices, and web services.

3. Agent Profile Model: this model specifies the profile of different context-aware agents, that is the type of services that an agent offers, the reason for acquiring certain contexts etc. Knowing these information, the broker can effective decide from which agents the true identity of users should be hidden, for what purpose an agent needs to access a user's personal information etc.

4. Information Privacy Model: this model specifies rules for determining whether a piece of contextual knowledge is sensitive user information and should it be treated with great care. This model does not exclude information that is not directly related to people to be sensitive information (e.g. the subject and the time of a meeting).

Domain Heuristics

Domain heuristics are rules for supporting context reasoning. The use of these rules is twofolded. First, heuristic rules can help the broker to resolve knowledge inconsistency that are caused by imperfect sensing. Second, heristic rules can help the broker to build and maintain a shared model of context.

User Privacy Policies

User privacy policies are rules that the broker and individual users have come to agree upon before the broker shares any of the user information with agents in the system. Each user has his/her own privacy policy with the broker. Each policy specifies who defines the policy, what information can or cannot be shared, what agents does a specific rule apply to etc.

3.4.2 Inference Engine

In each broker, the Knowledge Base is always coupled with an Inference Engine. The Inference Engine consists of three reasoning modules that each performs distinctive reasoning tasks. These modules are **Ontology Reasoning Module**, **Context Reasoning Module** and **Knowledge Maintenance Module**.

Ontology Reasoning Module

The main function of the Ontology Reasoning Module is to deduce facts that can be concluded from the knowledge in the Knowledge Base in conjunction with the models that are defined in the Domain Ontology. For example, based on the Context Model, the Ontology Reasoning Module will reason about the containment relationships between conference rooms and buildings; based on the Information Source Model, this module will conclude in order to acquire the location context of a meeting, the broker must communicate with the web service which maintains the schedule of the meeting; based on the Agent Profile Model, this model will deduce that any agent that does not carry a valid security certificate will not be allow to share any information of the user context; based on the Information Privacy Model, this module will conclude, the subject of the meeting and the identity of participants are all considered as sensitive information in a business board meeting.

Context Reasoning Module

The Context Reasoning Module of an Inference Engine is responsible for reasoning about the contexts of the domain. In contrast to the Ontology Reasoning Module which focuses on ontology reasoning, this module focuses on context reasoning. Context reasoning, again, is the process of reasoning about situational conditions of an entities by aggregating information that are acquired from the physical environment.

The underlying reasoning mechanism of this module may be a hybrid composition of logic reasoning (e.g. deduction, fuzz logic etc.) and statistical analysis (e.g. decision trees, Bayesian networks etc.). The reasoning procedures defined in this module will exploit the heuristic knowledge defined in the Domain Heuristic component (see Sec. 3.4.1).

Knowledge Maintenance Module

The Inference Engine of a broker also include a module called Knowledge Maintenance Module which is solely responsible for maintaining the consistency of contextual knowledge in the Knowledge Base. There are two different aspects of knowledge inconsistency. In one aspect, knowledge stored in the Knowledge Base may be inconsistent because there are noise in the information acquired from the physical environment. In another aspect, knowledge may be inconsistent because knowledge stored in the Knowledge Base does not accurately reflect the dynamic changes in the environment.

To maintain consistency of the Knowledge Base, this module will periodically monitor the facts that are stored in the Knowledge Base. When inconsistent knowledge exists, this module will attempt to resolve the inconsistency by modifying the Knowledge Base. There are two methods for detecting knowledge inconsistency:

1. Check logical implication: in this method, inconsistency are detected through a systematic process of checking logically implications between different facts and rules. For example, assume that the Knowledge Base has a rule which states "`If A is true, then B must be true`" and a fact which states "`B is not true`". Now, a new fact "`A is true`" is asserted, and which leads to the new conclusion of "`B is true`". At this moment, the module will detect that it is logically incorrect for both facts "`B is not true`" and "`B is true`" at the same time. Thus, the module concludes there is an inconsistency in the Knowledge Base.

2. Apply domain heuristics: in this method, inconsistency are detected through a reasoning process which attempt to maintain consistency between the current state of the world with the heuristics that defined in Domain Heuristics. For example, let's assume that there is a heuristic rule which state "`No one person can be in two different rooms at the same time`" and there is a fact which states "`Bob is currently attending a meeting in RM 201`" and another fact which states "`Bob is currently typing a paper on his computer in RM 102`". It is clear that both facts cannot be both true at the same time in according to the defined heuristic. Thus, the module concludes there is an inconsistency in the Knowledge Base.

3.4.3 Context Acquisition Component

In the COBRA architecture, brokers do not directly access low-level sensors for contexts. Context acquisition is facilitated by a middle-ware infrastructure call Context Acquisition Component. This component is similar to the widget infrastructure defined in the Context Toolkit (see discussion in Sec. 2.4.2). The design of the Context Acquisition Component has two layers: **Context Sensors** and **Context Interpreter**.

Context Sensors are sources of information from which contextual information can be acquired. Sensors include both hardware sensors and virtual sensors. For example, RFID badges are hardware sensors, and web services that provide meeting schedule information are virtual sensors. Including virtual sensors as a part of the context sensors is one of the difference between the architecture of Context Toolkit and the COBRA architecture.

Context Interpreter is an ontology driven inference engine which interprets the sensed information. Based on the Domain Ontology that are defined in the Knowledge Base, the Context Interpreter infers the contexts and asserts the results into the Knowledge Base. Context Interpreter does not perform any consistency checks when asserting contextual knowledge. It is the Inference Engine's responsibility to maintain knowledge consistency.

3.4.4 Broker Behavior

Broker Behavior defines how a broker should behave when interact with users and agents in the system. The behavior of the broker can be thought as a collection of protocols that the broker is obliged to follow during its interaction with users and agents. The Broker Behavior consists of two distinctive protocols, one for interacting with the users called **Privacy Policy Negotiation** and one for interacting with the agents called **Knowledge Sharing**.

Privacy Policy Negotiation

This protocol is executed when the broker attempts to establish a privacy policy agreement with a user before sharing any of his/her contextual information with other agents (Fig. 3.3 shows a UML sequence diagram of this protocol).

The negotiation between the broker and the user starts when the broker is requested by some agent to reveal a user's contexts and no privacy policy has been established with that user in the past. In the case, the broker first *proposes* a default privacy policy to the user. The default policy contains pre-defined policy rules for guiding broker's decisions in sharing user contexts. Upon receiving the proposal, the user may *accept* or *reject* the proposal. If the proposal is accepted, the broker will *inform* the agent of the requesting information. If the proposal is rejected, the broker will not reveal the context of user to the requesting agent. After rejecting the proposal, the user has the option to *counter propose* a modified version of the privacy policy. After receiving a proposed for modifying the privacy policy, the broker will re-initiate the negotiation protocol.

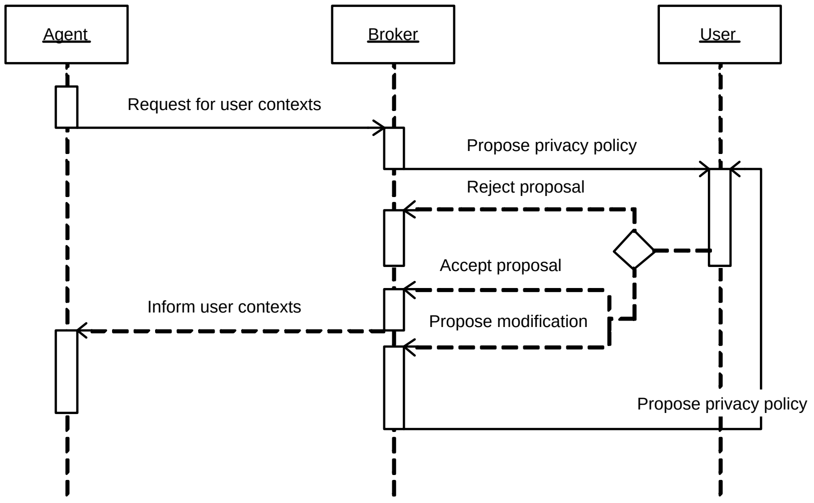

Figure 3.3: A UML diagram of the privacy policy negotiation protocol

Knowledge Sharing

In the COBRA architecture, any agents in the system can share knowledge with the broker. Agents can follow one of the three protocols to share knowledge with the broker: 1) inform protocol, 2) query protocol and 3) subscription protocol. It is important to note that when sharing information about user contexts, all three protocols will be executed in according to the policy rules that are defined by individual users.

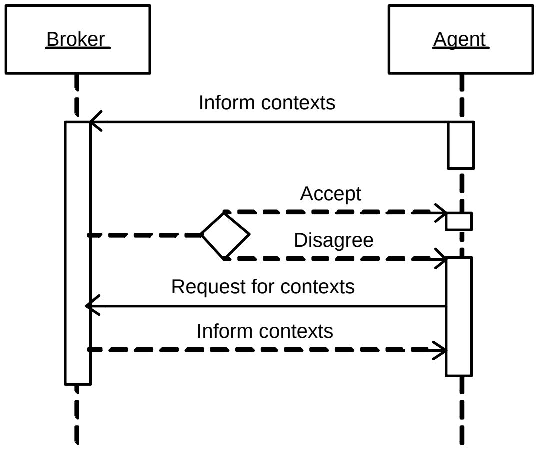

Figure 3.4: A UML diagram of the inform protocol

The inform protocol (see Fig. 3.4) begins when some agent wants to *inform* the broker of certain contextual knowledge. After receiving the inform message, the broker can either *accept* the contextual facts that are stated in the message or the broker can *disagree* with these facts (e.g. strong evidences in the Knowledge Base show that these facts are inaccurate). If the informing agent receives a disagreement message from the broker, the agent has the option to *request* broker's opinion of the context.

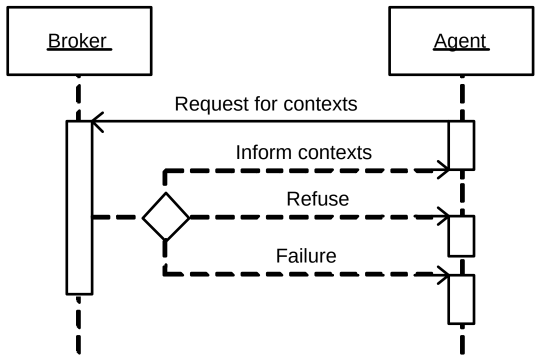

Figure 3.5: A UML diagram of the query protocol

The query protocol (see Fig. 3.5) begins when some agent wants to know about some contextual knowledge of the domain. The agent *request* some specific contextual knowledge from the broker. Upon receiving the request, the broker must decide if the requesting information is allowed to be shared in according user privacy policy rules. If not, the broker will send a *refuse* message to the agent, indicating the broker refuse to provide such information. If privacy policies allow requesting information to be shared, then the broker will check to see if the existing Knowledge Base has such information. If there are matching information, then the broker will *inform* the agent with the needed contextual knowledge. Otherwise, a *failure* message is replied, indicating the broker is unable to provide requesting contexts.

The subscription protocol (see Fig. 3.6) begins when some agent wants to be notified about certain context changes in the domain. In order to be notified about context changes, the agent *subscribe* an event change notification with the broker. Upon receiving the subscription, the broker first checks to see if the information described in the subscription is allowed by the user privacy policies. If not, the broker sends a *refuse* message to the requesting agent. Otherwise, an *agree* message is sent. When context changes occur, the broker's Knowledge Base is modified. In the case, if any context change subscription match the occurring context changes, then the broker will *notify* the senders of those subscriptions.

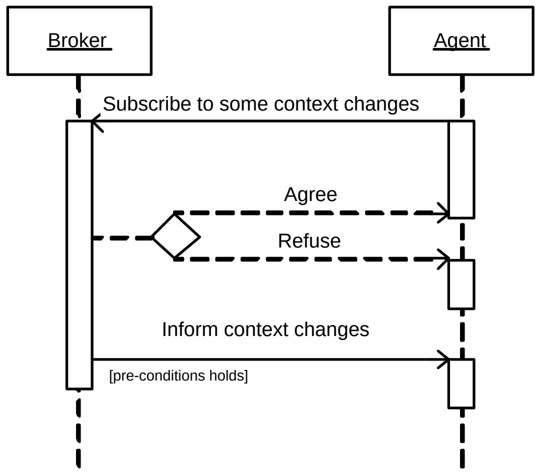

Figure 3.6: A UML diagram of the subscription protocol

Chapter 4

Summary and Research Plan

Building successful context-aware systems requires adequate supports from computing infrastructure at many different levels. If no infrastructure supports are provided to enable context-aware agents to share a common model of context or to allow users to control how their personal information is shared and used in a context-aware environment, building context-aware systems will not only be difficult but also costly.

In Ch. 2, I have reviewed the previous research work that have been done to provide architectural support for context-aware systems. Based on my review, none of the existing architectures adequately supports both sharing of contexts among resource-limited agents and protecting the privacy of users in a context-aware environment. In order to overcome these barriers, in Ch. 3, I have describes a preliminary design of Context Broker Architecture which is aimed to reduce the cost and difficulty of building context-aware agents in an Intelligent Meeting Room environment.

In the rest of this chapter, I will describe the research plan to complete my PhD. dissertation. First, I will discuss critical technologies that are essential for implementing the COBRA architecture. Second, I will describe the method to evaluate the feasibility of the architecture. Third, I will present the milestones of the future research.

4.1 Initial Approach to Implementation

4.1.1 Modeling Ontology and Policy using OWL

In order to study how context-aware agents can exploit the emerging Semantic Web technologies, I will use Web Ontology Language (OWL) to model various ontology and policy in COBRA. OWL is a semantic mark-up language for defining Web ontology [47]. In OWL, an ontology is a set of definitions of classes and properties. The ontology constraints on the way those classes and properties can be employed (Fig. 4.1 shows a partial example of an OWL ontology).

```
<owl:Class rdf:ID="TalkInSession"/>

<owl:ObjectProperty rdf:ID="talkContext"/>

<owl:ObjectProperty rdf:ID="hasTalkContext">
  <rdfs:domain rdf:resource="&tc;TalkInSession"/>
  <rdfs:range rdf:resource="&tc;talkContext"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasAudience">
  <rdfs:subPropertyOf rdf:resource="&tc;hasTalkContext"/>
  <rdfs:range rdf:resource="&tc;Audience"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Audience"/>
```

Figure 4.1: A partial example of an OWL ontology

When modeling COBRA ontologies, each ontology model (i.e. Context Model, Information Source Model and Agent Profile Model) will be defined in two parts: *base ontology* and *system ontology*. The base ontology covers the basic terminologies for describing concepts in the domain. Using OWL, the base ontology specifies what are the class of things and their associated properties that can be defined. Sharing base ontology will allow independently developed agents to communicate and share contextual knowledge. The system ontology, on the other hand, defines the specific concepts of the domains using the terminologies in the base ontology. In another words, it defines the instances of class of things in the domain.

When modeling user privacy policies, the base ontology defines the terminologies for describing policy statements. Each policy statement will consist of information about the user who define the policy, the terms and conditions that a broker has agreed to honor when sharing user contexts etc.. When defining concepts for modeling policy terms and conditions, some of the standard ontology for describing policy statements (e.g. P3P [16]) will be adopted.

4.1.2 Building the Inference Engine using Jess

When taking a Web ontology approach to represent and store knowledge, the knowledge base of the system must be coupled with a specialized inference engine. Jess [23], a rule-based expert system shell for the Java platform, will be used for building the Inference Engine of COBRA. The main advantage of using Jess is its interoperability with the existing Semantic Web programming libraries and tools that are written in Java. Because Jess is a rule-based engine completely implemented in Java, Jess can also interoperate with some of the popular agent development platforms (e.g. JADE/FIPA).

In order to develop a COBRA inference engine to process OWL ontology, specialized inference mechanisms must be coupled with Jess. An initial approach is to develop the inference mechanism to reason over OWL ontologies in *triples* [35]. In Jess syntax, a triple statement can be expressed as `triple([pred],[subj],[obj])`. The general idea to construct the reasoning mechanism is the following (the details of the reasoning mechanism is described in [31]):

1. Translate OWL statements to triples: define a set of rules to translate XML-encoded OWL ontology into triple statements in according to the semantics defined in OWL axioms [38]

2. Build rules from triples: define a set of rules to give domain specific interpretations to the triple statements in the knowledge base. These interpretations forms system reasoning rules (e.g. rules to reason about contexts, rules to reason about permission to share user contexts etc.).

There are few challenges in building an inference engine to work with triple statements. First, a standard set of first-order axioms for interpreting OWL is not currently available. This can be a problem for systems that do not adopt the same logical interpretations to interoperate. Second, when building a hybrid reasoning mechanism (i.e. integrating fuzz logic, decision tree etc. to form layers of context reasoning), the triple representation of knowledge may not always be suitable and efficient. Third, although OWL provides language features for detecting conflicts and inconsistence, at present no off-the-shelf inference engine or programming library are available to exploit these features.

4.1.3 Implementing the Broker Behavior using JADE

The COBRA architecture is aimed to provide an engineering solution for building context-aware agents in an Intelligent Meeting Room environment. To construct a concrete COBRA system, I will exploit FIPA standards and a FIPA-compliant agent system called JADE [5]. The main advantage of following the FIPA agent paradigm is that FIPA provides a rich collection of software standards for building interoperable agent systems in large scale. Based on the FIPA standards, JADE can greatly simplifies the development of distributed agent systems and can offer opportunities for running FIPA agents on mobile devices such as Pocket PC and GSM cell phones [7].

When building a COBRA system using JADE, the life cycle of agents and the communication between agents will follow the semantics of FIPA standards. From JADE's perspective, a Domain Context Broker is simply a regular agent. However, from COBRA's perspective, a broker is a persistent agent service that will have the same life cycle as the other standard services such as the Agent Management Service (AMS) and the Directory Facilitator (DF). When a COBRA system starts up, the brokers of each domain register their services with the system's DF. Each COBRA system defines its own ontology for describing agent services. When an agent joins the system, the agent can lookup the brokers of different domains by querying the DF. The communication between the agent and the brokers will be facilitated by the underlying Agent Message Transport [22].

Although the JADE/FIPA programming infrastructure provide a foundation for building COBRA systems, a number of issues remain to be resolved. First, the protocols for agents to communicate with the brokers need to be defined for each Broker Behavior. For the Knowledge Sharing behavior, protocols can extend the existing FIPA's *inform* communicative act and the *subscribe* interaction protocol. For the Privacy Policy Negotiation behavior, however, new protocols will need to be developed. Second, message content specifications for each communication protocol also need to be defined, such as the content messages for describing queries and event notifications. Third, in a dynamic environment, communication security between the brokers and the agents are of great concern. In order address this problem, some of the emerging security infrastructure [29] for Ubiquitious Computing need to be studied and re-examined in context of COBRA.

4.2 Feasibility Study

In order to show that the difficulty and cost of building of context-aware systems can be greatly reduced by the use of the COBRA architecture, I will evaluate the feasibility of using the architecture in a prototype system. This prototype systems will be developed based on the Intelligent Meeting Room scenario described in Sec. 1.3.1. This system will provide context-aware services based on conference room activities in an acadimia environment.

This Intelligent Meeting Room system will consist of one domain which is called the conference room domain. The broker of this domain will be preconfigured with necessary sensors and background knowledge to manage and control the context model of the domain. This context model consists of information about 1) who is currently in the room, 2) what devices are currently in the room, 3) what is the current location of a person (if the person is not currently in the room), 4) what activities is currently going on in the room (i.e. meeting, presentation, class, not activity, or unknown), 5) what role does each person have in the current activity (i.e. audience, speaker, meeting organizer, student, or lecturer), and 6) what do each person intend to do.

In addition, the broker will be configured to acquire information from the following heterogenous sources: 1) the Semantic web sites that publish class schedule, talk schedule, and meeting schedule information, 2) the personal agents that run on users' web sites, 3) physical sensors for detecting people and device presence (i.e. RFID badge), and 4) device agents that are present in the conference room.

Thru a series of experiments in building this systems, I will attempt to validate the following hypothesises:

1. By exploiting the Semantic Web infrastructure, context-aware agents can access a wider range of contexts in addition to what can be already accessed through physical sensors.

2. By developing a hybrid context reasoning mechanism, we can help capability-limited agents to maintain a coherent and consistent model of contexts in a dynamic environment.

3. By enabling agents to share a common model of contexts, we will allow agents to access contexts that are otherwise inaccessible if they act in isolation.

4. By defining a policy-driven mechanism to control how users' personal contexts are used and shared, we can effective protect the privacy of users in a context-aware environment.

4.3 Milestones

The proposed research work is expected to be completed in a 12-18 months period. The work will be divided into 5 stages as follows:

- **Stage 1 (estimated 1-2 months)**: In this stage, I will develop a design specification for the Intelligent Meeting Room system based on the Context Broker Architecture. This specification will describe the service functions and behaviors of context-aware agents, and how they will interact with the broker in the domain.

- **Stage 2 (estimated 3-5 months)**: In this stage, I will prototype a Domain Context Broker to support context-aware agents. The prototype system will include implementations of the Knowledge Base, the Inference Engine, the Context Acquisition Component and Broker Behavior.

- **Stage 3 (estimated 3-5 months)**: In this stage, the main focus is to build a running demo system based on the COBRA architecture. I will implement the Intelligent Meeting Room system in according to the design specification developed in Stage 1.

- **Stage 4 (estimated 2-3 months)**: In this stage, I will evaluate the feasbility of the COBRA architecture by conducting a series of experiments based on the prototyped system.

- **Stage 5 (estimate 2-3 months)**: In this stage, the main focus is to complete the writing of the PhD. dissertation. Minor development efforts are expected, mainly in debugging and fine-tuning the context-aware agents and the COBRA architecture.

Bibliography

[1] Gregory D. Abowd, Anind K. Dey, Robert Orr, and Jason A. Brotherton. Context-awareness in wearable and ubiquitous computing. In *ISWC*, pages 179–180, 1997.

[2] Mark Ackerman, Trevor Darrell, and Daniel J. Weitzner. Privacy in context. *Special Issue on Context-Aware Computing. Human-Computer Interaction*, 16(2-4), 2001.

[3] Philip E. Agre. Changing places: Contexts of awareness in computing. *Special Issue on Context-Aware Computing. Human-Computer Interaction*, 16(2-4), 2001.

[4] Abhaya Asthana, Mark Cravatts, and Paul Krzyzanowski. An indoor wireless system for personalized shopping assistance. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[5] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Developing multi agent systems with a fipa-compliant agent framework. *Software - Practice And Experience*, 31(2):103–128, 2001.

[6] Frazer Bennett, Tristan Richardson, and Andy Harter. Teleporting - Making Applications Mobile. In *Proceedings of 1994 Workshop on Mobile Computing Systems and Applications*, Santa Cruz, December 1994.

[7] Federico Bergenti and Agostino Poggi. Leap: a fipa platform for handheld and mobile devices. In *Agent Theories, Architectures, and Languages (ATAL-2001)*, 2001.

[8] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, may 2001.

[9] Dan Brickley and R.V. Guha. Rdf vocabulary description language 1.0: Rdf schema. www.w3c.org, nov 2002.

[10] R. Brooks. The intelligent room project, 1997.

[11] Paul Castro and Richard Muntz. Using context to assist in multimedia object retrieval, 1999.

[12] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College, Computer Science, Hanover, NH, nov 2000.

[13] Harry Chen and Sovrin Tolia. Steps towards creating a context-aware agent system. Technical report, Hewlett Packard Labs, 2001.

[14] Harry Chen, Sovrin Tolia, Craig Sayers, Tim Finin, and Anupam Joshi. Creating context-aware software agents. In *Proceedings of the First GSFC/JPL Workshop on Radical Agent Concepts*, 2001.

[15] Michael H. Coen. Building brains for rooms: Designing distributed software agents. In *AAAI/IAAI*, pages 971–977, 1997.

[16] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle. The platform for privacy preferences 1.0 (p3p1.0) specification. www.w3c.org, jan 2002.

[17] Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Web ontology language (owl) reference version 1.0. W3C Working Draft – www.w3c.org, nov 2002.

[18] Michael Dertouzos. *The Unfinished Revolution.* HarperCollins Publishers, Inc., 2001.

[19] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis, Georgia Institute of Technology, 2000.

[20] Anind K. Dey, Jennifer Mankoff, and Gregory D. Abowd. Distributed mediation of imperfectly sensed context in aware environments.

[21] Merrian-webster online. www.m-w.com.

[22] Fipa abstract architecture specification. www.fipa.org.

[23] Ernest J. Friedman-Hill. *Jess, The Expert System Shell for the Java Platform.* Sandia National Laboratories, version 6.1a4 edition.

[24] Krzysztof Gajos. Rascal - a resource manager for multi agent systems in smart spaces. In *Proceedings of CEEMAS 2001*, 2001.

[25] Nicholas Hanssens, Ajay Kulkarni, Rattapoom Tuchinda, and Tyler Horton. Building agent-based intelligent workspaces. In *Proceedings of ABA Conference*, June 2002. To Appear.

[26] Beverly L. Harrison, Kenneth P. Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze me, hom me, tilt me! an exploration of manipulative user interfaces. In *CHI98*, 1998.

[27] Ken Hinckley, Jeffrey S. Pierce, Mike Sinclair, and Eric Horvitz. Sensing techniques for mobile interaction. In *UIST*, pages 91–100, 2000.

[28] Jason I. Hong and James A. Landay. An infrastructure approach to context-aware computing.

[29] Lalana Kagal, Tim Finin, and Anupam Joshi. Developing secure agent systems using delegation based trust management. In *Proceedings of Security of Mobile Multi-Agent Systems Workshop (AAMAS 2002*, 2002.

[30] Tim Kindberg and John Barton. A Web-based nomadic computing system. *Computer Networks (Amsterdam, Netherlands: 1999)*, 35(4):443–456, 2001.

[31] Joe Kopena and William C. Regli. Daml-jesskb: A tool for reasoning with semantic web. edge.mcs.drexel.edu/assemblies/software/damljesskb/articles/DAMLJessKB-2002.pdf.

[32] Sanjeev Kumar, Philip R. Cohen, and Hector J. Levesque. The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. In *Proceedings of the Fourth International Conference on Multi-Agent Systems*, pages 159–166, 2000.

[33] Mik Lamming and Mike Flynn. Forget-me-not: intimate computing in support of human memory. In *Proceedings FRIEND21 Symposium on Next Generation Human Interfaces*, 1994.

[34] Marc Langheinrich, Friedemann Mattern, Kay Rmer, and Harald Vogt. First steps towards an event-based infrastructure for smart things. Ubiquitous Computing Workshop at PACT 2000.

[35] Ora Lassila and Ralph R. Swick. Resource description framework (rdf) model and syntax specification. www.w3c.org, feb 1999.

[36] John McCarthy and Sasa Buvac. Formalizing context (expanded notes). In Sasa Buvač and Łucia Iwańska, editors, *Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language*, pages 99–135, Menlo Park, California, 1997. American Association for Artificial Intelligence.

[37] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *Sixteen ACM Symposium on Operating Systems Principles*, pages 276–287, Saint Malo, France, 1997.

[38] Peter F. Patel-Schneider, Patrick Hayes, Ian Horrocks, and Frank van Harmelen. Web ontology language (owl) abstract syntax and semantics. www.w3c.org, jul 2002.

[39] Filip Perich. A service for aggregating and interpreting contextual information. Technical report, Hewlett Packard Labs, 2002.

[40] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *Mobile Computing and Networking*, pages 32–43, 2000.

[41] Jun Rekimoto. Tilting operations for small screen interfaces. In *ACM Symposium on User Interface Software and Technology*, pages 167–168, 1996.

[42] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, pages 434–441, 1999.

[43] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[44] William Noah Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.

[45] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde. Advanced interaction in context. *Lecture Notes in Computer Science*, 1707:89–??, 1999.

[46] A. Schmidt and M. Beigl. There is more to context than location: Environment sensing technologies for adaptive mobile user interfaces, 1998.

[47] Michael K. Smith, Deborah McGuinness, Raphael Volz, and Chris Welty. Web ontology language (owl) guide version 1.0. www.w3c.org, nov 2002.

[48] Roy M. Turner. Context-sensitive reasoning for autonomous agents and cooperative distributed problem solving. In *Proceedings of the IJCAI Workshop on Using Knowledge in its Context*, Chambéry, France, 1993.

[49] Roy Want, Andy Hopper, Veronica Falcao, and Jon Gibbons. The active badge location system. Technical Report 92.1, Olivetti Research Ltd., ORL, 24a Trumpington Street, Cambridge CB2 1QA, 1992.

[50] Marc Weiser. The computer for the 21st century. *Scientific American*, 265(30):94–104, 1991.

[51] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. HTTP://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.h (Hypertext version of Knowledge Engineering Review paper), 1994.