A Middleware for Context-Aware Agents in Ubiquitous Computing Environments*

Anand Ranganathan and Roy H. Campbell

Department of Computer Science
University of Illinois at Urbana-Champaign, USA
{ranganat,rhc}@uiuc.edu

Abstract. Ubiquitous Computing advocates the construction of massively distributed systems that help transform physical spaces into computationally active and intelligent environments. The design of systems and applications in these environments needs to take account of heterogeneous devices, mobile users and rapidly changing contexts. Most importantly, agents in ubiquitous and mobile environments need to be context-aware so that they can adapt themselves to different situations. In this paper, we argue that ubiquitous computing environments must provide middleware support for context-awareness. We also propose a middleware that facilitates the development of context-aware agents. The middleware allows agents to acquire contextual information easily, reason about it using different logics and then adapt themselves to changing contexts. Another key issue in these environments is allowing autonomous, heterogeneous agents to have a common semantic understanding of contextual information. Our middleware tackles this problem by using ontologies to define different types of contextual information. This middleware is part of Gaia, our infrastructure for enabling Smart Spaces.

1 Introduction

Ubiquitous Computing Environments consist of a large number of autonomous agents that work together to transform physical spaces into smart and interactive environments. In order for an agent to function effectively in these environments, they need to perform two kinds of tasks – they need to sense and reason about the current context of the environment; and they need to interact smoothly with other agents. In this paper, we propose a middleware for Ubiquitous Computing Environments that meets these two needs of agents in the environment.

The role of context has recently gained great importance in the field of ubiquitous computing. "Context" is any information about the circumstances, objects, or conditions by which a user is surrounded that is considered relevant to the interaction between the user and the ubiquitous computing environment [1]. A lot of work has been done in trying to make applications in ubiquitous computing environments context aware so that they can adapt to different situations and be more receptive to users' needs[1][2][3][8][13].

Humans behave differently in different contexts. They are able to sense what their context is and they adapt their behavior to their current context. The way humans

* This research is supported by a grant from the National Science Foundation, NSF CCR 0086094 ITR and NSF 99-72884 EQ

M. Endler and D. Schmidt (Eds.): Middleware 2003, LNCS 2672, pp. 143–161, 2003.

adapt themselves is based on rules that they learn over the course of their experiences. Humans are, thus, able to follow socially and politically correct behavior that is conditioned by their past experiences and their current context.

Automated agents (which may be applications, services and devices) too, can follow contextually-appropriate behavior, if they are able to sense and reason about the context in which they are operating. Ubiquitous computing environments are characterized by many sensors that can sense a variety of different contexts. The types of contexts include physical contexts (like location, time), environmental contexts (weather, light and sound levels), informational contexts (stock quotes, sports scores), personal contexts (health, mood, schedule, activity), social contexts (group activity, social relationships, other people in a room), application contexts (email received, websites visited) and system contexts (network traffic, status of printers)[9]. Agents in these environments should be able to acquire and reason about these contexts to adapt the way they behave.

In this paper, we argue that ubiquitous computing environments must provide middleware support for context awareness. A middleware for context awareness would provide support for most of the tasks involved in dealing with context. Context-aware agents can be developed very easily with such a middleware. A middleware for context-awareness would also place different mechanisms at the disposal of agents for dealing with context. These mechanisms include reasoning mechanisms like rules written in different types of logic (first order logic, temporal logic, fuzzy logic, etc.) as well as learning mechanisms (like Bayesian networks, neural networks or reinforcement learning). Developers of context-aware agents would not have to worry about the intricate details of getting contextual information from different sensors or developing reasoning or learning mechanisms to reason about context.

Another important requirement of middleware in ubiquitous computing environments is that they allow autonomous, heterogeneous agents to seamlessly interact with one another. While a number of protocols and middlewares (like TCP/IP, CORBA, Jini, SOAP, etc.) have been developed to enable distributed agents to talk to one another, they do not address the issues of syntactic and semantic interoperability among agents. They do not provide a common terminology and shared set of concepts that agents can use when they interact with each other. This problem is especially acute in the realm of contextual information since different agents could have different understandings of the current context. They might use different terms to describe context, and even if they use the same terms, they might attach different semantics to these terms. A middleware for context-awareness must address this problem by ensuring that there is no semantic gap between different agents when they exchange contextual information.

We have identified several requirements for a middleware for context-awareness in ubiquitous computing environments. These are:

1. Support for gathering of context information from different sensors and delivery of appropriate context information to different agents.
2. Support for inferring higher level contexts from low level sensed contexts
3. Enable agents use different kinds of reasoning and learning mechanisms
4. Facilities for allowing agents to specify different behaviors in different contexts easily.
5. Enable syntactic and semantic interoperability between different agents (through the use of ontologies)

In this paper, we propose a middleware for promoting context-awareness among agents in ubiquitous computing environments. This middleware is based on a predicate model of context. The model of context and the middleware also supports the use of different reasoning mechanisms like first order logic and temporal logic by agents to reason about context and decide how to behave in different contexts. Agents can alternatively employ learning mechanisms like Bayesian learning and reinforcement learning to learn different behaviors in different contexts. Different logics have different power, expressiveness and decidability properties. Agents can choose the appropriate logic that best meets their reasoning requirements.

The middleware uses ontologies to define the semantics of various contexts. The ontologies define the structure and the properties of different types of contextual information. They allow different agents in the environment to have a common semantic understanding of different contexts.

Ontologies have been used extensively in the Semantic Web[14] to allow semantic interoperability among different web-based agents. DAML+OIL[20] has emerged as one of the premier languages for describing ontologies in the Semantic Web. Our ontologies are also written in DAML+OIL. The use of standard technologies for semantics allows semantic interoperability between agents in our environment and other external agents (in other environments or on the web). The use of ontologies, thus, dramatically increases the scalability of the environment.

Our middleware allows rapid prototyping of context-aware agents in ubiquitous computing environments. It also allows agents the use of powerful reasoning mechanisms to handle contextual information and ensures syntactic as well as semantic interoperability between different agents through the use of ontologies. The middleware has made it very easy for us to develop a variety of context-aware applications and services.

In the rest of the paper, we describe how our middleware achieves context awareness in and semantic interoperability between agents in ubiquitous computing environments. In Section 2, we provide motivation for middleware support for context-awareness. In Section 3, we describe our predicate model for context, which forms the basis of the various reasoning and learning mechanisms that we use. Section 4 introduces Gaia, our infrastructure for Smart Spaces, into which our middleware for context awareness and semantic interoperability has been integrated. Section 5 describes how context awareness has been achieved among agents. Section 6 describes the use of ontologies in the middleware. Section 7 describes our current implementation status; Section 8 has related work; Section 9 has future work and Section 10 concludes the paper.

2 Why a Middleware for Context-Awareness?

Different approaches have been suggested for promoting context-awareness among agents. Anind Dey et al[1] have proposed the Toolkit approach, which provides a framework for the development and execution of sensor-based context-aware applications and provides a number of reusable components. The toolkit supports rapid prototyping of certain types of context-aware applications.

The other approach is developing an infrastructure or a middleware for context-awareness. A middleware would greatly simplify the tasks of creating and maintain-

ing context-aware systems[2]. A middleware would provide uniform abstractions and reliable services for common operations. It would, thus, simplify the development of context-aware applications. It would also make it easy to incrementally deploy new sensors and context-aware agents in the environment. A middleware would be independent of hardware, operating system and programming language. Finally, a middleware would also allow us to compose complex systems based on the interactions between a number of distributed context-aware agents.

While traditional middleware like CORBA and Jini do provide the basic mechanisms for different objects (or agents) to communicate with each other, they fall short in providing ways for agents to be context aware. Context-awareness involves acquisition of contextual information, reasoning about context and modifying one's behavior based on the current context. A middleware for context-awareness would provide support for each of these tasks. It would also define a common model of context, which all agents can use in dealing with context. It would also ensure that different agents in the environment have a common semantic understanding of contextual information.

Our middleware for context-awareness does use CORBA to enable distributed agents to find and communicate with one another. It, however, provides extra functionality to enable context-awareness. It is based on a predicate model of context and uses ontologies to describe different types of contexts. It also provides various services and libraries to enable agents acquire and reason about contextual information easily.

3 Context Model

In order to allow applications to be context-aware, we first need a model for context. We have developed a context model that is based on predicates. We use ontologies to describe the properties and structure of different context predicates. This context model provides the basis for reasoning about contexts using various mechanisms.

3.1 The Context Predicate

We represent a context as a predicate. We follow a convention where the name of the predicate is the type of context that is being described (like location, temperature or time). This convention allows us to have a simple, uniform representation for different kinds of contexts. Besides, it also allows us to easily describe the different contexts in an ontology, as we shall see later. It is also possible to have relational operators like "=" and "<" as arguments of a predicate.

Example context predicates are:

- Location (chris , entering , room 3231)
- Temperature (room 3231 , "=" , 98 F)
- Sister(venus , serena)
- StockQuote(msft , ">" , $60)
- PrinterStatus(srgalw1 printer queue , is , empty)
- Time(New York , "<" , 12:00 01/01/01)

The values that the arguments of a predicate can take are actually constrained by the type of context. For example, if the type of context is "location", the first argument has to be a person or object, the second argument has to be a preposition or a verb like "entering," "leaving," or "in" and the third argument must be a location. We perform type-checking of context predicates to make sure that the predicate does make sense.

3.2 Ontologies to Describe Context Predicates

The structures of different context predicates are specified in an ontology[15]. Each context type corresponds to a class in the ontology. This ontology defines various context types as well as the arguments that the predicates must have. The ontology is written in DAML+OIL[20], which is fast becoming the de-facto language of the semantic web[14].

For example, many context predicates are defined to have arguments in an SVO (Subject Verb Object) format. Thus, the structure of these predicates is ContextType(<Subject>,<Verb>,<Object>). For instance, the ontology declares that the Location predicate must have a subject which belongs to the set of persons or things, a verb or preposition like "inside" or "entering" and a location, which may be a room or a building.

The ontology is used to check the validity of context predicates. It also makes it easier to write different context predicates since we know what the structure of the predicate is and what kinds of values different arguments can take. It also allows different ubiquitous computing environments to inter-operate since it is possible to define translations between the terms used in the ontologies of these environments. Section 6 has more details about the use of ontologies.

This logical model for context is quite powerful. It allows us to describe the context of a system in a generic way, which is independent of programming language, operating system or middleware. Since the structure and the semantics of context predicates are specified in an ontology, it allows different components in the system to have a common understanding of the semantics of different contexts.

The predicate model of context is also generic enough to allow different reasoning mechanisms. For example, it is possible to write rules using these context predicates that describe application behavior using different logics like first order logic or temporal logic. It is also possible to perform other kinds of inferencing based on these predicates using Bayesian networks, neural networks or other approaches.

4 Gaia

Our middleware for context awareness and semantic interoperability has been integrated into Gaia[16][17]. Gaia is our infrastructure for Smart Spaces, which are ubiquitous computing environments that encompass physical spaces. The main aim of Gaia is to make physical spaces like rooms, homes, buildings and airports intelligent, and aid humans in these spaces. Gaia converts physical spaces and the ubiquitous computing devices they contain into a programmable computing system. It offers services to manage and program a space and its associated state. Gaia is similar to

traditional operating systems in that it manages the tasks common to all applications built for physical spaces. Each space is self contained, but may interact with other spaces. Gaia provides core services, including events, entity presence (devices, users and services), discovery and naming. By specifying well defined interfaces to services, applications may be built in a generic way so that they are able to run in arbitrary active spaces. The core services are started through a bootstrap protocol that starts the Gaia infrastructure. Gaia uses CORBA to enable distributed computing.

Gaia consists of a number of different types of agents performing different tasks. There are agents that perform various core services required for the functioning of the environment like discovery, context-sensing, event distribution, etc. There are agents associated with devices that enable them be a part of the environment. Each user also has an agent that keeps personal information and acts as his proxy in a variety of settings. Finally there are application agents that help users perform various kinds of tasks in the environment. Examples of application agents include PowerPoint applications, music playing applications and drawing applications.

5 Enabling Context-Awareness

The Gaia middleware provides different ways for agents to acquire various types of contextual information and then reason about it. A diagram of our infrastructure for context-awareness is shown in Fig 1.

5.1 Overview of Context Infrastructure

There are different kinds of agents that are involved in the Context Infrastructure within Gaia (Fig. 1). These are:

- *Context Providers*. Context Providers are sensors or other data sources of context information. They allow other agents (or Context Consumers) to query them for context information. Some Context Providers also have an event channel where they keep sending context events. Thus, other agents can either query a Provider or listen on the event channel to get context information.
- *Context Synthesizers*. Context Synthesizers get sensed contexts from various Context Providers, deduce higher-level or abstract contexts from these simple sensed contexts and then provide these deduced contexts to other agents. For example, we have a Context Synthesizer which infers the activity going in a room based on the number of people in the room and the applications that are running.
- *Context Consumers*. Context Consumers (or Context-Aware Applications) are agents that get different types of contexts from Context Providers or Context Synthesizers. They then reason about the current context and adapt the way they behave according to the current context.
- *Context Provider Lookup Service.* Context Providers advertise the context they provide with the Context Provider Lookup Service. This service allows agents to find appropriate Context Providers. There is one such service in a single ubiquitous computing environment.

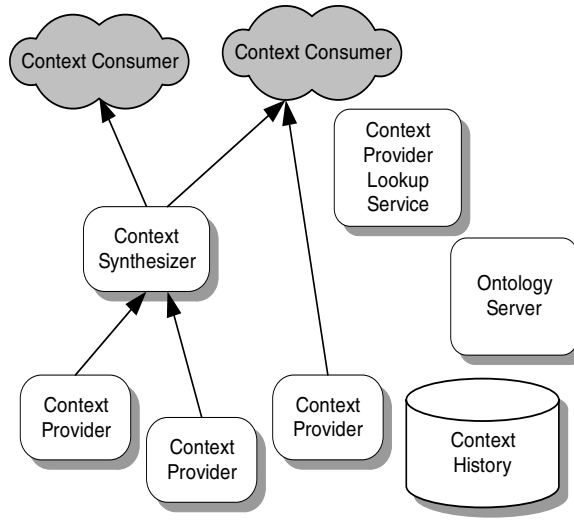

Fig. 1. Gaia Context Infrastructure

- *Context History Service.* Past contexts are logged in a database. The Context History Service allows other agents to query for past contexts. There is one such service in a single ubiquitous computing environment.
- *Ontology Server.* The Ontology Server maintains ontologies that describe different types of contextual information. There is one Ontology Server per ubiquitous computing environment.

These different kinds of agents are described in further detail in the following sections.

5.2 Use of Different Reasoning Mechanisms by Different Agents

A key feature of our middleware is that it endows agents with a variety of reasoning and/or learning mechanisms to help them reason about context appropriately. Using these reasoning or learning mechanisms, agents can infer various properties about the current context, answer logic queries about context or adapt the way they behave in different contexts.

Agents can reason about context using rules written in different types of logic like first order logic, temporal logic, description logic, higher order logic, fuzzy logic, etc. Different agents have different logic requirements. Agents that are concerned with the temporal sequence in which various events occur would need to use some form of temporal logic to express the rules. Agents that need to express generic conditions using existential or universal quantifiers would need to use some form of first order logic. Agents that need more expressive power (like characterizing the transitive closure of relations) would need higher order logics. Agents that deal with specifying terminological hierarchies may need description logic. Agents that need to handle uncertainties may require some form of fuzzy logic.

Instead of using rules written in some form of logic to reason about context, agents can also use various machine learning techniques to deal with context. Learning techniques that can be used include Bayesian learning, neural networks, reinforcement learning, etc. Depending on the kind of concept to be learned, different learning mechanisms can be used. If an agent wants to learn the appropriate action to perform in different states in an online, interactive manner, it could use reinforcement learning or neural networks. If an agent wants to learn the conditional probabilities of different events, Bayesian learning is appropriate. The decision on what kind of logic or learning mechanism to use depends not only on the power and expressivity of the logic, but also on other issues like performance, tractability and decidability.

Our middleware provides agents a choice of reasoning and learning mechanisms that they can use to understand and react to context. Our current implementation allows reasoning based on many-sorted first order logic, propositional linear-time temporal logic or probabilistic propositional logic. It also allows agents to learn using Bayesian methods or through reinforcement learning. These mechanisms are provided in the form of libraries that the agent can use. We discuss the power, expressivity and decidability of these logics in the implementation section. In the following sections, we describe how Context Providers, Context Synthesizers and Context Consumers use various reasoning mechanisms to perform their tasks.

5.3 Context Providers

Context Providers sense various types of contexts and allow these contexts to be accessed by other agents. We have a number of Context Providers in our infrastructure providing various types of contexts like location, weather, stock price, calendar and schedule information, etc.

Different Context Providers use different reasoning or learning mechanisms for reasoning about the contexts they sense and for answering queries. For example, Context Providers that deal with uncertain contexts could use fuzzy logic, while those that require the ability to quantify over variables could use first order logic. Our Location Context Provider, for instance, uses first order logic so that it can quantify over people or over locations. It can thus answer queries concerning all the people in a room, or all the locations in a building. Our Weather Forecast Context Provider uses a form of fuzzy logic to attach probabilities with different contexts. For instance, it says that precipitation could occur with a certain probability the next day.

Context Providers provide a query interface for other agents to get the current context. Depending on the type of logic or learning mechanism used, Context Providers have different ways of evaluating queries. However, all reasoning and learning mechanisms are based on the predicate model of context, which is defined in the ontology. So, in spite of different Providers using different logics, their common grounding on the predicate model makes it easy for Context Consumers to query them in a uniform way.

The query interface is similar to that of Prolog. If the query is a predicate with no variables, then the result is expected to be the truth value of that predicate. The result is, thus, either a “yes” or a “no” (or a probability of the context predicate being true). If the query is a predicate with variables, then the result must include any unifications of the variable with constants that make the predicate true (if there are any). The re-

sulting unified context predicates that are returned may have additional attributes like time or probability depending on the type of logic used.

Some Context Providers (like those that provide dynamic contexts and are associated with sensors) also send events about their context on an event channel. Consumers can listen on this channel. For example, our room-based location service sends an event like "*Location(Bob, Entering, Room 3231)*" when Bob enters Room 3231. Exactly when a Context Provider generates an event is set by a policy for the Context Provider. In some cases, the provider keeps sending events periodically. For example, our weather service keeps sending temperature updates every 5 minutes. In other cases, the provider sends an event whenever a change in context is detected.

All Context Providers support a similar interface for getting contexts and listening to context events. So, consumers don't have to worry about the actual type of Context Provider they are querying. This greatly aids development of context-aware applications.

5.4 Context Synthesizers

Context Synthesizers are agents that provide higher-level contexts based on simpler sensed contexts. A Context Synthesizer gets source contexts from various Context Providers, applies some sort of logic to them and generates a new type of context. A Context Synthesizer is both a Context Provider and a Context Consumer. Just like Context Providers, Context Synthesizers also support a Prolog-like query interface which other agents can use to get the current context. They may also send events in an event channel.

We follow two basic approaches for inferring new contexts from existing contexts. The first uses static rules to deduce higher-level contexts and the other uses machine learning techniques.

Rule Based Synthesizers. Rule-based synthesizers use pre-defined rules written in some form of logic to infer different contexts. For example, we have a Room Activity Context Provider that based on the number of people in the room and the applications running in the room deduces what kind of activity is going on in the room. It uses rules written in first order logic to perform the deduction. Some of the rules that this Room Activity Context Provider employs are:

1. #People(Room 2401, ">=" , 3) AND Application(PowerPoint, Running) => RoomActivity(2401, Presentation)
2. #People(Room 2401, ">=" , 1) AND Application(MPEG Player, Running) => RoomActivity(2401, Movie Screening)
3. #People(Room 2401, ">=" , 3) AND NOT $\exists_{\text{Entertainment-Application}}$ x Application(x, Running) => RoomActivity(2401, Meeting)
4. #People(Room 2401, "=" , 1) AND Application(Visual Studio, Running) => RoomActivity(2401, Individual Development)
5. #People(Room 2401, "=" , 2) AND Application(Visual Studio, Running) => RoomActivity(2401, Extreme Programming)
6. #People(Room 2401, "=" , 0) => RoomActivity(2401, Idle)

Each of the rules also has a priority associated with it – so if more than one rule is true at the same time, exactly what the activity in the room is determined using the priorities of the rules. If two rules are true at the same time and they have the same priority, then one of them is picked at random.

Even the context concerning the number of people in the room is an inferred context. So, the Room Activity Context Provider has to keep track of these entered and exited context events and infer the number of people in the room based on these events. Whenever the Room Activity Context Provider deduces a change in the activity in the room, it sends an event with the new activity.

We found that a fairly small set of rules are sufficient for deducing the activity in a room in most circumstances. The Room Activity Context Provider did accurately deduce the activity in the room most of the times. This proves, at least empirically, that rule-based synthesizers are fairly useful in deducing some types of contexts.

The middleware provides mechanisms for developers to specify the rules of inference for these Synthesizers very easily. The developer can browse the ontology to get the terminology used in the environments. He can then make use of this terminology to frame the rules. The middleware also abstracts away many tasks like getting references to appropriate Context Providers, querying them or listening to their event channels and sending context events at appropriate times. The developer is thus free to concentrate on the task of writing the rules.

Synthesizers that Learn. Rule based synthesizers have the disadvantage that they require explicit definition of rules by humans. They are also not flexible and can't adapt to changing circumstances. Making use of machine learning techniques to deduce the higher-level context enables us to get around this problem.

One type of context that is extremely difficult to sense is the mood of a user. It is difficult to write rules for predicting user mood since each user is different. There are such a large number of factors that can influence a user's mood. We attempted to use a learning mechanism that took some possible factors into account like his location, the time of day, which other people are in the room with him, the weather outside and how his stock portfolio is faring.. Our User Mood Context Provider uses the Naïve Bayes algorithm for predicting user mood. We make use of past contexts to train the learner. During the training phase, we ask the user for his mood periodically. We construct a training example by finding the values of the features (ie. location, weather, etc.) for each time the user entered his mood. We train the learner for a week. Once the training phase is over, the learner can predict the mood of the user given the values of the feature contexts (which are represented as predicates). The result of a Naïve Bayes algorithm is probabilistic. It would be possible to retain the probabilities associated with different user moods and thus some form of fuzzy logic or probabilistic reasoning in handling these contexts. However, we just consider the mood with the highest probability and assume that to be true.

We found that this user mood predictor did predict the moods of user fairly well in different situations after some training. Humans are fairly repetitive creatures –their moods in different contexts follow certain predictable patterns. Of course, the predictions are not always perfect – we can only make good guesses based on the information we have available. It is quite difficult to take into account all possible factors that can influence the mood of a user.

The middleware aids the training and the actual operation of the agent. It provides support for many tasks like getting references to appropriate Context Providers, querying them or listening to their event channels and sending context events at appropriate times. The developer is thus free to concentrate on the task of learning.

5.5 Context Consumers (or Context-Aware Applications)

Context Consumers are agents that consume various types of contexts and adapt their behavior depending on the current context. As mentioned earlier, consumers can obtain contexts by either querying a Context Provider or by listening for events that are sent by Context Providers. Our middleware makes it very easy to develop and deploy context aware applications. It is easy for applications to get the contexts they require to make decisions. In our infrastructure, Context Consumers get references to Context Providers using the Context Provider Lookup Service.

Specifying Context-Sensitive Behavior. One common way in which applications can be made context sensitive is to specify actions to be performed whenever the context of the environment changes. Thus, whenever the context of the environment changes, the application reconfigures itself to meet the requirements of the new context. For example, a jukebox application in a smart room may reconfigure itself whenever a person enters or leaves the room by changing the song it is playing, the volume of the song or the speakers it uses to play the music. This application model is based on the ECA (event-condition-action) execution model[24].

Our context framework provides a number of ways for developers to specify different behaviors in different contexts. Just as in the case of Context Synthesizers, there are two broad ways in which these behaviors can be described. The first is to allow application developers to write rules that indicate what actions are to be performed in different contexts. The second is to use machine learning approaches that learn what actions to perform in different contexts.

Rule-Based Approaches. Using rules to specify application behavior is a very simple way to make applications context-sensitive. These rules consist of conditions and actions. Whenever the context of the environment changes, the conditions in all the rules are evaluated. If any of the conditions become true, then the actions corresponding to these rules are evaluated. Each rule is also associated with a certain priority, which is used in case there is a conflict in the actions.

The conditions in the rules are expressions in some form of logic like first order logic, temporal logic, description logic, higher order logic, fuzzy logic, etc. Our framework for context awareness is flexible enough to allow the use of any form of logic for writing rules. Depending on the kind of logic used to express the condition, different evaluation engines are used to decide whether a condition is true or false.

The middleware makes it very easy to develop rule-based Context Consumers. These agents have a configuration file that lists all the rules. Actions are specified as methods in the agent that are invoked when the context becomes true. The middleware provides support for getting references to appropriate Context Providers, getting context information from them, evaluating the rules and invoking appropriate methods in different contexts. It also makes available different evaluation engines in the

form of libraries, which aid in the reasoning process. The job of the agent developer is, thus, very simple. He can concentrate on writing the rules that govern agent behavior without worrying about other things.

A sample configuration file for a jukebox application agent is shown in Table 1. This agent plays appropriate music in a room depending on who is in the room, what the weather is outside and how the stock portfolio of the user is faring. The rules of this agent are written in first order logic.

In our current implementation, developers of context-aware applications need to write such a configuration file (using the appropriate kind of logic) for describing behavior in different contexts. However, we are also working on a graphical interface, which simplifies the developer's task. This graphical interface would show the various types of contexts available (as defined in the ontology), allow the developer to construct complex rules involving these contexts in different types of logics and also present him with a list of possible behaviors of the application for these different contexts.

Another example, which uses temporal logic, is a slideshow agent. One of its rules is that it starts playing a particular ppt file on a large plasma screen when Chris enters the room and continues playing until the Activity in the room is Meeting. This rule is written as:

```
Condition: Location(Chris, Entered, 2401) AND (TRUE UNTIL Ac-
tivity(2401, Meeting))

Action: PlayOnPlasmaScreen("scenery.ppt")

Priority: 2
```

Machine Learning Approaches. The disadvantage of using rule-based approaches for developing context sensitive applications is that they are not flexible and cannot adapt themselves to changing circumstances. Use of machine learning techniques helps us get around this problem. Developers need not specify the behavior of applications in different scenarios; the application can learn the most appropriate behavior in different contexts.

A variety of machine learning techniques can be employed for learning appropriate behavior. These include Bayesian approaches, neural networks, Support Vector Machines, various clustering algorithms, reinforcement learning, etc.

Learning can take place either in a batch-processing or in an online fashion. Batch-processsing approaches (like Naïve Bayes, etc.) require a number of training examples. Gaia stores all events that are sent in the environment in a database. These include events with context information, events that describe user and application actions, etc. The stored past events act as training examples for our learner. This approach is especially useful for learning user behavior by studying his actions over a period of time. If user behavior is learned well, applications can take proactive actions on behalf of the user depending on the context and thus save the user's valuable time.

Online learning mechanisms (like reinforcement learning) can learn their concepts while operating in the environment. These mechanisms involve trying out different actions, observing the user's reaction to these actions and learning which actions are better in different situations. For example, we have developed an intelligent Notification Service that tries to learn the most appropriate times to send different types of notifications. It can send different types of notifications like stock prices, weather

Table 1. First Order Rules for deducing activity in a room

```
ThereExists(Person) x Location(x, Entered, 2401).
PlayWelcomeMessage().
Priority:1.

Location(Manuel, Entered, 2401) OR Location(Chris, Entered,
2401).
ShowInterface().
Priority:1.

Location(Manuel,In, 2401).
PlayRockMusic().
Priority:1.

Location(Manuel,In,2401)AND Temperature(Champaign,>,50)
PlaySoftMusic()
Priority:2

Location(Bhaskar,In, 2401) AND Location(Chris,In, 2401)
PlayPopMusic()
Priority:2

Location(Bhaskar,In, 2401) AND Location(Chris,In, 2401) AND
Temperature(Champaign,>,50)
PlayHiphopMusic()
Priority:4

Location(Bhaskar,In, 2401) AND
StockPrice(MSFT,>,50)
PlayHappyMusic()
Priority:2
```

information, news headlines and error messages on different kinds of media like on tickertape, by speech or by email. The Notification Service learns by sending a notification in some situation and observing the user's reaction to the notification. The user can give feedback about the notification by rating its usefulness (or even by stopping the notification midway). Depending on the feedback of the user, the Notification Service either increases or decreases the probability that it would send the same type of notification in a similar situation again.

As in the case of rules, the middleware provides support for getting references to appropriate Context Providers, getting context information from them, evaluating the concept learned and invoking appropriate actions in different contexts. It also provides libraries and utilities that aid the learning process.

5.6 Context Provider Lookup Service

The Context Provider Lookup Service allows searches for different context providers. Providers advertise the set of contexts they provide with the Context Provider Lookup Service. This advertisement is in the form of a first order expression. Agents can query the Lookup Service for a context provider that provides contextual information it needs. The Lookup Service checks if any of the context providers can provide what the agent needs and returns the results to the application.

For example, a location context provider that tracks Bob's location around the building advertises itself as $\forall_{Location} y$ *Location(Bob, In, y)*. An application that wants to know when Bob enters room 3231, would send the query *Location(Bob, In, Room 3231)*to the Lookup Service. The Lookup Service sees that the context provider does provide the context that the application is interested in (the advertisement is a superset of the query) and returns a reference to the context provider to the application.

5.7 Context History

Applications can make use of not just the current context, but also past contexts to adapt their behavior for better interacting with users. We thus store contexts continuously, as they occur, in a database. The Gaia event service allows event channels to be "persistent", ie. all events sent on these channels are stored in a database along with a timestamp indicating when the event was sent.

It is thus possible to store all context events (or a certain subset of them) in a database. Since all context events have a well-determined structure (as given by the ontology), it is relatively simple to automatically develop schemas for storing them into a database. Storing past contexts enables the use of data mining to learn and discover patterns in user behavior, room activities and other contexts. This sort of data mining can, for example, be used in security applications like intrusion detection, where any observed behavior way outside the ordinary can be construed as an intrusion.

6 Ontologies for Semantic Interoperability

Ubiquitous Computing Environments feature a large number of autonomous agents. Various types of middleware (based on CORBA, Java RMI, SOAP, etc.) have been developed that enable communication between different entities. However, existing middleware have no facilities to ensure semantic interoperability between the different entities. Since agents are autonomous, it is infeasible to expect all of them to attach the same semantics to different concepts on their own. This is especially true for context information, since different agents could have a different understanding of the current context and can use different terms and concepts to describe context.

In order to enable semantic interoperability between different agents, we take recourse to methods used in the Semantic Web[14]. Ontologies establish a joint terminology between members of a community of interest. These members can be humans or automated agents. Each agent in our environment uses the vocabulary and concepts defined in one or more ontologies. When two different agents talk to each other, they know which ontology the other agent uses and can thus understand the semantics of what the other agent is saying.

Another advantage of using standard technologies employed in the Semantic Web for describing semantics is scalability. Since agents in our environment use ontologies described in the same language (DAML+OIL) as those on the web, we enable semantic interoperability between our agents and other external agents (in other environments or on the web).External agents can refer to ontologies used in our environment while interacting with our agents. They can, thus, find out what terms and concepts are used by our agents and communicate meaningfully with them. Similarly, agents in our environment can refer to the ontologies used by external agents while communi-

cating with them. The use of ontologies, thus, enables agents in different ubiquitous computing environments to have a common vocabulary and a common set of concepts while interacting with one another.

6.1 Ontologies in Gaia

We have developed ontologies for describing various concepts in a Ubiquitous Computing Environment. We have ontologies that describe the different kinds of agents and their properties. These ontologies define different kinds of applications, services, devices, users, data sources and other agents. They also define all terms used in the environment and the relationships between different terms. They establish axioms on the properties of these agents and terms (written in Description Logic) that must always be satisfied.

We also have ontologies that define the structure of contextual information. These are useful for checking the validity of context information. They also makes it easier to specify the behavior of context-aware applications since we know the types of contexts that are available and their structure. They ensure that all agents in the system have the same semantic understanding of different pieces of contextual information.

6.2 The Ontology Server

All the ontologies in Gaia are maintained by an Ontology Server. Other agents in Gaia contact the Ontology Server to get descriptions of agents in the environment, meta-information about context or definitions of various terms used in Gaia. It is also possible to support semantic queries (for instance, classification of individuals or subsumption of concepts). Such semantic queries require the use of a reasoning engine that uses description logics like the FaCT reasoning engine[21]. We plan on providing support for such queries in the near future.

The Ontology Server also provides an interface for adding new concepts to existing ontologies. This allows new types of contexts to be introduced and used in the environment at any time. The Ontology Server ensures that any new definitions are logically consistent with existing definitions.

The use of ontologies also makes it possible for agents in different environments to inter-operate. To support such an inter-operation, mappings need to be developed between concepts defined in the ontologies of the two environments. We plan on developing a framework for supporting such inter-operation very soon.

6.3 Ontologies for Smoother Interaction between Agents

Since the ontologies clearly define the structure of contextual information, different agents can exchange different types of context information easily. Context Consumer agents can get the structure of contexts they are interested in from the Ontology Server. They can then frame appropriate queries to Context Providers to get the contexts they need.

Context Providers and Context Synthesizers can also get the structure of contexts that they provide from the Ontology Server. So, they know the kinds of queries they

can expect. They also know the structure of events that they need to send on event channels.

Finally, ontologies also help the developer when he is writing rules or developing learning mechanisms for context aware agents. The developer has access to the set of terms and concepts that describe contextual information. He can thus use the most appropriate terms and concepts while developing context aware agents.

7 Implementation

All agents in the Context Infrastructure are implemented on top of CORBA and are a part of Gaia. This means they can be instantiated in any machine in the system, can access event channels, can be moved from one machine to another and can be discovered using standard mechanisms like the CORBA Naming Service and the CORBA Trading Service. The Context History Service uses the MySQL database for storing past contexts.

We currently support a number of reasoning mechanisms including many sorted first order logic and linear time propositional temporal logic. For reasoning in first order logic, we use XSB[19] as the reasoning engine. XSB is a more powerful form of Prolog which uses tabling and indexing to improve performance and also allows limited higher order logic reasoning. We use the many-sorted logic model where quantification is performed only over a specific domain of values. The ontology defines various sets of values (like Person, Location, Stock Symbol, etc). Thus, the Person set consists of the names of all people in our system. The Location set consists of all valid locations in our system (like room numbers and hallways). Stock Symbol consists of all stock symbols that the system is interested in (e.g. IBM, MSFT, SUNW, etc.). Each of these sets is finite. Quantification of variables is done over the values of one of these sets. Since quantification is performed only over finite sets, evaluations of expressions with quantifications will always terminate. More discussion on the issues of decidability and expressiveness can be found in [10][11][12].

For temporal logic, we have developed our own reasoning engine that is based on Templog[23]. Templog is a logic programming language, similar to Prolog, which allows the use of temporal operators. We restrict the power of this logic to propositional logic, which makes it decidable and also simpler to evaluate.

We also currently support some machine learning mechanisms, viz. Naïve Bayes learning and reinforcement learning. The Naïve Bayes approach involves learning conditional probabilities between different events from a large number of training examples. Thus given a certain context, it gives the conditional probability that some action should be performed or some other context should be true. The reinforcement learning approach involves trying to learn appropriate actions based on user feedback.

An ontology of all terms used in the context infrastructure has been developed in DAML+OIL. The Ontology Server uses the FaCT reasoning engine[21] for checking the validity of context expressions.

We have implemented a number of Context Providers in our system such as providers of location, weather, stock price, calendar contexts and authentication contexts. We also have some context synthesizers as described earlier. Some examples are a Synthesizer which deduces the mood of a user using Naïve Bayes learning; and another which deduces the activity in the room using rules. The middleware has allowed us to develop a number of context aware applications very easily. Some context-

aware applications we have developed are a context-sensitive jukebox, a context-sensitive chat application[18] and a context-sensitive notification service.

One of the main features of our middleware is that it greatly helps in the development of context-aware applications. The benefits of using the middleware include reduced development times of context-aware applications and great ease in specifying complex behaviors of these applications. Developers do not have to worry about the details of getting contextual information from different sources or the mechanics of triggering different actions in different situations. This helps in rapid development and prototyping of applications.

The middleware also makes it pretty simple to insert new sensors and new Context Synthesizers, which infer different contexts, into the system. Since all the terms used in the environment are defined in the ontology, it is easy to frame rules for inferring contexts based on these terms. The developer does not have to worry about not using inappropriate terms or concepts, since he can refer to the definitions in the ontology when in doubt.

8 Related Work

A lot of work has been done in the area of context-aware computing in the past few years. Seminal work has been done by Anind Dey, et al. in defining context-aware computing, identifying what kind of support was required for building context aware applications and developing a toolkit that enabled rapid prototyping of context-aware applications[1]. While the Context Toolkit does provide a starting point for applications to make use of contextual information, it does not provide much help on how to reason about contexts. It does not provide any generic mechanism for writing rules about contexts, inferring higher-level contexts or organizing the wide range of possible contexts in a structured format.

In [2], Jason Hong, et. al., make the distinction between a toolkit and an infrastructure. An infrastructure, according to Hong, is a well-established, pervasive, reliable set of technologies providing a foundation for other systems. Our middleware for context-awareness builds on Hong's notion of an infrastructure and provides a foundation for developing context-aware applications easily.

Bouquet, et al. [4] address the problem of contexts in autonomous, heterogeneous distributed applications, where ach entity has its own notion of context depending on its viewpoint. To interact with other entities, an entity should know the relationship between its viewpoint and other entities' viewpoint. Our middleware uses ontologies to achieve this inter-operability in a more generic fashion. Paul Castro and his colleagues [5] have worked on developing "fusion services" which extract and infer useful context information from sensor data using Bayesian networks. Our middleware provides a more generic framework where such learning approaches can be used.

Terry Winograd compares different architectures for context[6] and proposes one that uses a centralized Event Heap[7]. Our system, however, provides a framework where distributed reasoning can take place. In [3], Brumitt, et al describe their experiences with multi-modal interactions in context-aware environments and how such an environment can respond automatically to different contexts. Our middleware provides an easy way for developers to specify how an environment should automatically respond to different contexts.

Reconfigurable Context-Sensitive Middleware [22] provides context-sensitive applications with adaptive object containers (ADCs) for runtime context data acquisition, monitoring and detection. Applications can specify behavior using a context-aware IDL. Our middleware provides a more generic way of specifying the behavior of context-aware applications using different reasoning and learning mechanisms.

9 Future Work

There are a number of possible enhancements to our middleware. A new approach to developing context-sensitive applications is by modeling them as state machines. This allows their behavior to be determined by specific sequences of context changes. State machine approaches to modeling applications are useful especially when a sequence of changes in context needs to trigger a sequence of actions by the application.

We have not yet tackled the issues of privacy and security. Some context information may be private and hence, all agents may not have access to them. The ontology can potentially encode such privacy and security constraints. It can thus be used to ensure that the rules developed for applications do not violate security restrictions.

We are also working on better user interfaces for developing context-aware applications, so that any ordinary user can program his or her own context-aware agent. These interfaces can make use of the ontologies to get the structures of different types of contexts and thus allow the user to develop rules with context information.

One aspect which we haven't studied as yet is the usability of context-aware applications. How will ordinary users deal with applications that try to learn their behaviors and their preferences? Will users take the time to write rules for specifying context-sensitive behavior of applications? Will users respond positively to the fact that the behavior of applications can change according to the context, and is hence not as predictable as current applications?

10 Conclusion

In this paper, we have described our middleware for developing context-aware applications. The middleware is based on a predicate model of context. This model enables agents to be developed that either use rules or machine learning approaches to decide their behavior in different contexts. The middleware uses ontologies to ensure that different agents in the environment have the same semantic understanding of different context information. This allows better semantic interoperability between different agents, as well as between different ubiquitous computing environments. Our middleware allows rapid prototyping of context-sensitive applications. We have developed a number of context-sensitive agents on our middleware very easily.

References

1. Dey, A.K., et al. "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", anchor article of a special issue on Context-Aware Computing, Human-Computer Interaction (HCI) Journal, Vol. 16, 2001.
2. Hong, J. I., et al. "An Infrastructure Approach to Context-Aware Computing". HCI Journal, '01, Vol. 16

3. Shafer, S.A.N., et al. "Interaction Issues in Context-Aware Interactive Environments." Special issue on Context-Aware Computing, Human-Computer Interaction (HCI) Journal, Vol. 16, 2001.
4. Bouquet, P., et al. "Context-Aware Distributed Applications" IRST Technical Report 0101-04, Instituto Trentino di Cultura, January 2001
5. Castro, P., et al. "Managing Context for Internet Video Conferences: The Multimedia Internet Recorder and Archive". Multimedia and Computer Networks 2000, San Jose, CA, January 2000
6. Winograd T. "Architectures for Context" In Human-Computer Interaction (HCI) Journal, '01, Vol. 16.
7. Johanson, B., et al. "The Event Heap: An Enabling Infrastructure for Interactive Workspaces" https://graphics.stanford.edu/papers/eheap/
8. Pascoe, J., et al. "Issues in Developing Context-Aware Computing" Proceedings of the International Symposium on Handheld and Ubiquitous Computing, Sept. 1999, Springer-Verlag, pp. 208-221.
9. Korkea-aho, M. "Context-Aware Applications Survey", http://www.hut.fi/~mkorkeaa/doc/context-aware.html
10. Shmueli O., "Decidability and expressiveness aspects of logic queries", Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of database systems, March 23 - 25, 1987, San Diego, CA USA , pp 237 - 24
11. Chandra, A.K., et al. "Horn Clauses Queries and Generalization", J Logic Programming 1985
12. Jarke, M., et al. "An Optimizing PROLOG Front-End to a Relational Query System", in Proceedings of ACM SIGMOD '84 Conference, pp296-306, Boston, MA, June 1984
13. Schilit, W. N., "A Context-Aware System Architecture for Mobile Distributed Computing", PhD Thesis, Columbia University, May 1995.
14. Berners-Lee T., et al. "A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities" http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html
15. Guarino N. "Formal Ontology in Information Systems" Proc. of FOIS'98, Trento, Italy
16. Román, M., et al, "Gaia: A Middleware Infrastructure to Enable Active Spaces". In *IEEE Pervasive Computing*, pp. 74-83, Oct-Dec 2002..
17. Hess, C.K.,et al, "Building Applications for Ubiquitous Computing Environments" In *International Conference on Pervasive Computing (Pervasive 2002)*, pp. 16-29, Zurich, Switzerland, August 26-28, 2002.
18. Ranganathan, A., et al, "ConChat: A Context-Aware Chat Program" . In *IEEE Pervasive Computing*, pp. 52-58, July-Sept 2002.
19. XSB – http://xsb.sourceforge.net
20. Harmelon, F., et al " Reference Description of the DAML+OIL ontology markup language", http://www.daml.org/2001/03/reference.html
21. Horrocks, I., "The FaCT System", Automated Reasoning with Analytic Tableaux and Related Methods, 1998
22. Yau, S., et al, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing". In *IEEE Pervasive Computing*, pp. 33-40, July-Sept 2002.
23. Abadi, M. et al. "Temporal Logic Programming" Journal of Symbolic Computation, 8:277-295, 1989
24. Dayal, U., et al. "The Architecture of an Active Database Management System". ACM SIGMOD Conference 1989, pp 215-224