An Ontology for Context-Aware Pervasive Computing Environments*

Harry Chen, Tim Finin, and Anupam Joshi
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
{hchen4, finin, joshi}@cs.umbc.edu

Abstract

This document describes COBRA-ONT an ontology for supporting pervasive context-aware systems. COBRA-ONT, expressed in the Web Ontology Language OWL, is a collection of ontologies for describing places, agents, events and their associated properties in an intelligent meeting room domain. This ontology is developed as a part of the Context Broker Architecture (CoBrA), a broker-centric agent architecture that provides knowledge sharing, context reasoning, and privacy protection supports for pervasive context-aware systems. We also describe an inference engine for reasoning with information expressed using the COBRA-ONT ontology and the ongoing research in using the DAML-Time ontology for context reasoning.

1 Introduction

Computing is moving toward pervasive, ubiquitous environments in which devices, software agents, and services are all expected to seamlessly integrate and cooperate in support of human objectives – anticipating needs, negotiating for service, acting on our behalf, and delivering services in an anywhere, any-time fashion [12]. An important next step for pervasive computing is the integration of intelligent agents that employing knowledge and reasoning to understand the local context and share this information in support of intelligent applications and interfaces. We are developing a new architecture called Context Broker Architecture (CoBrA) to support context-aware systems in smart spaces (e.g., intelligent meeting rooms, smart homes, and smart vehicles).

Context-aware systems are computer systems that can provide relevant services and information to users by exploiting context [4]. By context, we mean information about a location, its environmental attributes (e.g., noise level, light intensity, temperature, and motion) and the people, devices, objects and software agents it contains. Context may also include system capabilities, services offered and sought, the activities and

*This work was partially supported by DARPA contract F30602-97-1-0215, NSF award 9875433, NSF award 0209001, and Hewlett Packard.

tasks in which people and computing entities are engaged, and their situational roles, beliefs, and intentions.

We believe ontologies are key requirements for building context-aware systems for the following reasons: (i) a common ontology enables knowledge sharing in an open and dynamic distributed systems [20], (ii) ontologies with well defined declarative semantics provide a means for intelligent agents to reason about contextual information, and (iii) explicitly represented ontologies allow devices and agents not expressly designed to work together to interoperate, achieving "serendipitous interoperability" [15].

In CoBrA we have defined a collection of ontologies called COBRA-ONT for modeling the context in an intelligent meeting room environment. COBRA-ONT expressed in the Web Ontology Language OWL [1] defines typical concepts associated with places, agents, and events. To reason over knowledge that is described in COBRA-ONT and the OWL language, we have prototyped F-OWL an ontology inference engine, using the Flora-2 system [25] in XSB [22].

The rest of this document is organized as the following: in the next section we discuss the shortcomings of the previous pervasive context-aware systems that do not make explicit representation of context. An overview of the CoBrA system is given in the Section 3. In Section 4 we present COBRA-ONT and its associated use cases. An overview of the F-OWL inference engine is described in the Section 5. Future work and conclusions are given in the Section 6 and 7, respectively.

2 Shortcomings of the Previous Systems

A number of computing systems developed in the past aim to support pervasive computing (e.g., the Intelligent Room [7, 6], Cooltown [16], and Context Toolkit [23]). While these systems have made progress in various aspects of pervasive computing, they offer only weak support for knowledge sharing and context reasoning. A significant source of this weakness is that they are not built on a foundation of common ontologies with explicit semantic representation [5]. For example, in previous systems [3, 8, 24], user location information is widely used for guiding the adaptive behavior of the systems. However, none have taken advantage of the semantics of spatial relations in reasoning about context (i.e., information that describes the whole physical space that surrounds a particular location and its relationship to other locations).

Furthermore, previous systems often implemented context as simple programming language objects (e.g., Java class objects) or informally described in documentation. Because these representations require the establishment of a priori low-level implementation agreement between of the programs that wish to share information, they cannot facilitate knowledge sharing in an open and dynamic environment. In order to facilitate the sharing of contextual knowledge, we believe ontologies of context related information must be defined in order to provide a set of common vocabularies with shared semantics.

3 Context Broker Architecture

CoBrA is a broker-centric agent architecture for supporting context-aware systems in smart spaces. Central to our architecture is the presence of an intelligent agent called the *context broker*. The context broker is a specialized server entity that runs on a resource-rich stationary computer in the space (e.g., on a Mocha PC[1]). In a smart space, the context broker's role is to maintain a shared model of context on the behalf of a community of agents and devices in the space and to protect the privacy of users by enforcing the user-defined policies when sharing information with agents in the associated space.

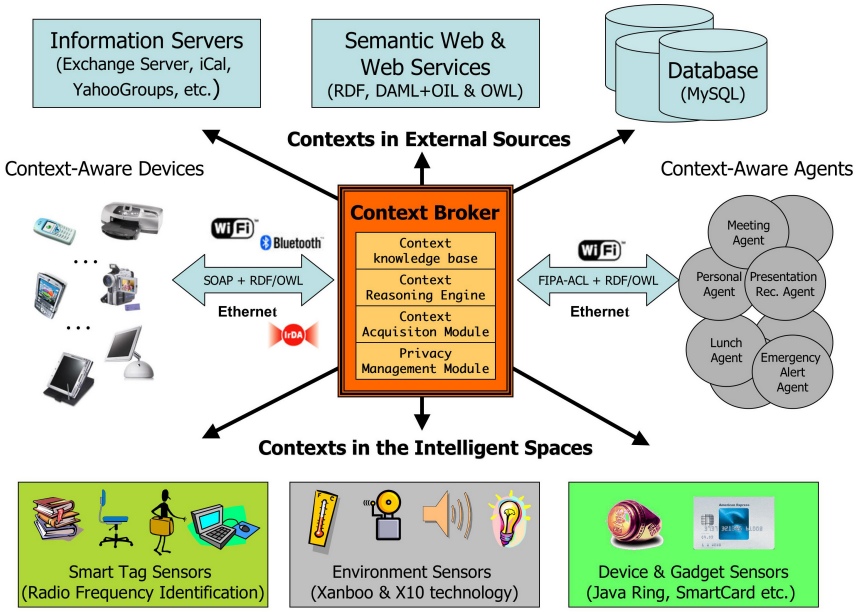

Figure 1: An intelligent context broker acquires context information from devices, agents and sensors in its environment and fuses it into a coherent model, which is then shared with the devices and their agents.

Figure 1 shows the high-level design of the broker and its relationship with other agents in a smart space. All computing entities in a smart space are presumed to have priori knowledge about the presence of a context broker, and the high-level agents are presumed to communicate with the broker using the standard FIPA Agent Communication Language [13]. The design of the context broker comprises the following four functional components: (i) **Context Knowledge Base**: a persistent storage of the context knowledge, (ii) **Context Reasoning Engine**: a reactive inference engine that reasons over the stored context knowledge, (iii) **Context Acquisition Module**: a library

[1] `http://www.cappuccinopc.com/mochap4.asp`

of procedures that form a middle-ware abstraction for context acquisition, (iv) **Policy Management Module**: a set of inference rules that deduce instructions for deciding the right permissions for different computing entities to share a particular piece of contextual information and for selecting the recipients to receive notifications of context changes.

Our centralized broker design addresses two important issues that are key to realizing the potential of pervasive computing: *supporting resource-limited mobile computing devices* and *addressing the concerns for user privacy*. With the introduction of a context broker that operates on a stationary computer, the burdens of acquiring and reasoning over contextual information is shifted away from the resource-limited mobile devices to the resource-rich broker; the complications inherent in establishing, monitoring and enforcing security, trust, and privacy polices are simplified in the presence of a centralized manager.

Although the existence of a context broker brings about advantages, the centralized design of a broker could create a "bottle neck" situation in a large scale intelligent space such as a building or a university campus, hindering the overall system performance. To address this problem, we propose a fault-tolerance approach, based on the *persistent broker team* design described by Kumar and Cohen [18]. In our design, multiple brokers are grouped together to form a *broker federation*. Each broker in the federation is responsible for managing a part of the intelligent space (e.g., a particular room in a building). In a federation, brokers are organized according to some communication structure (e.g., peer-to-peer or hierarchical), and they periodically exchange and synchronize contextual knowledge. A key advantage of this approach is that the access to a shared model of context is no longer solely depend on the availability of a single broker, and members of a broker federation share the repressibility of other members to provide access to the shared model of context.

4 An Overview of COBRA-ONT

Ontologies play an important role in CoBrA, helping the context broker to share contextual knowledge with other agents and enabling it to reason about context. COBRA-ONT is a collection of ontologies expressed in the Web Ontology Language OWL for describing information in an intelligent meeting room environment.

The Web Ontology Language OWL, which is similar to the other Semantic Web languages such RDF [19], RDFS [2], and DAML+OIL [9], is a language for publishing and sharing ontologies. We have chosen the OWL language to model context ontologies for the following reasons:

- OWL is much more expressive than RDF or RDFS, allowing us to build more knowledge into the ontology. For example, cardinality constraints can be imposed on the properties of an OWL class. In defining the location property of a person class, cardinality constraints can be used to restrict the number of physical location that a person can possibly be in at a given time instant (e.g., restricting the maximum cardinality to one means only one physical location instance is allowed to be associated with the location property of a person).

- OWL was expressly designed as an ”ontology language” and has many predefined classes and properties useful for expressing information about ontologies. For example, an ontologiy can import other ontogies, committing to all of their classes, properties and constraints. There are properties for asserting or denying the equivalence of individuals and classes, providing a way to relate information expressed in one ontology to another. These features, along with many others, are important for supporting ontology reuse, mapping and interoperability.

- OWL has been designed as a standard and has the backing of a well known and regarded standard organization (i.e. W3C). For this reason, there is a wide variety of development tools available for integrating the OWL ontologies into the development of our software applications.

4.a COBRA-ONT Use Cases

The development of COBRA-ONT focuses on creating ontologies that are suitable for building pragmatic context-aware systems. Some typical use cases of COBRA-ONT are the following:

- A sensor agent detects the presence of a Bluetooth-enabled cellphone in Room 210. It composes a description of this sensed event using COBRA-ONT, which then is sent to the context broker in the associated space. Without having any evidence to the contrary, the broker asserts that the owner the device is also in present in Room 210. Based on a physical location ontology predefined in COBRA-ONT, knowing Room 210 is a part of the Computer Science Building which in turn is a part of the UMBC campus, the context broker concludes the device owner is in school today.

- After a speaker enters the meeting room, her mobile device sends the context broker her predefined user policy, which describes the privacy rules that the broker should enforce while she is attending the meeting. Knowing the user does not want to reveal her home address to services at the meeting, based on a privacy protection ontology predefined in COBRA-ONT, the broker reasons that it should keep secret of her home phone number also since it’s relatively easy to determine an address given a telephone number.

- After a talk given by a distinguished professor, a student from the audience takes few pictures of the speaker. Before his digital camera sends the pictures to the photo album agent at home, it checks with the broker to ensure that there are no prohibitions on publishing photographs of the event, acquires the location and event information from the context broker and embeds that information into the pictures’ meta-data. Upon receiving the pictures, based on the shared location and event ontologies, the photo album agent reason about the context in which the pictures are taken and automatically archive them into the appropriate albums in his photo library.

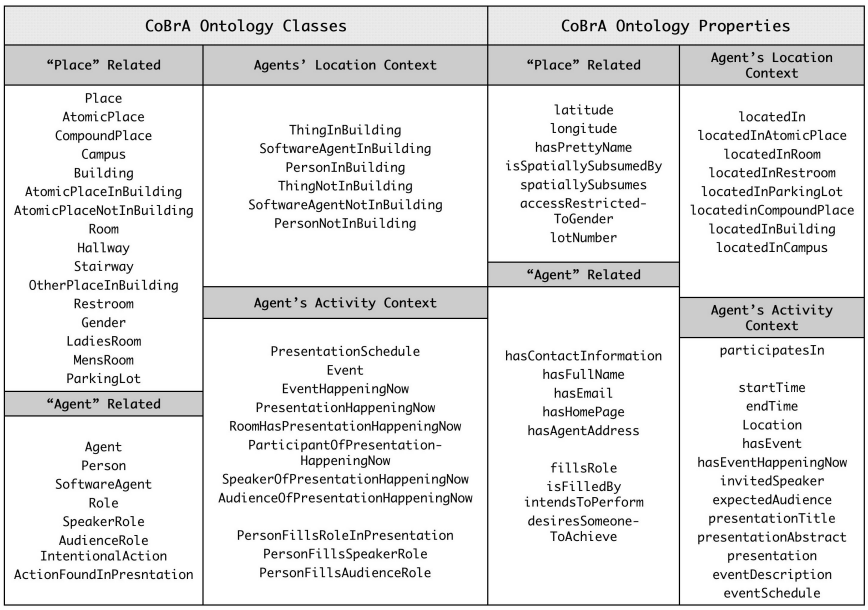

Figure 2: A complete list of the classes and properties in COBRA-ONT v0.2.

4.b Key Concepts in COBRA-ONT

We describe version 0.2 of the COBRA-ONT ontology[2]. Figure 2 shows a complete list of the classes and properties in COBRA-ONT, which consists of 41 classes (i.e., RDF resources that are type of `owl:class`) and 36 properties (i.e., RDF resources that are type of either `owl:ObjectProperty` or `owl:DatatypeProperty`).

Our ontology is categorized into four distinctive but related themes: (i) ontologies about physical places, (ii) ontologies about agents (both human and software agents), (iii) ontologies about the location context of the agents, and (iv) ontologies about the activity context of the agents.

4.b.1 Ontologies about Places

A top level class in COBRA-ONT is `Place`, which represents the abstraction of a physical location. It has a set of properties that are typically used to describe a location (e.g., longitude, latitude, and string name). COBRA-ONT defines two special subclasses called `AtomicPlace` and `CompoundPlace` to represent two different classes of the physical locations that have distinctive containment property (see Figure 3). The containment property of a physical location is defined as its model for being

[2]A complete version of the ontology is available at `http://daml.umbc.edu/ontologies/cobra/0.2/cobra-ont`. Future versions of the ontology can be accessed through `http://cobra.umbc.edu`

capable of spatially subsuming other physical locations. For example, in our ontology, a campus spatially subsumes all buildings on the campus, and a building spatially subsumes all rooms that are in it.

```
<owl:Class rdf:ID="Place">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AtomicPlace"/>
    <owl:Class rdf:about="#CompoundPlace"/>
  </owl:unionOf>
  …
</owl:Class>

<owl:Class rdf:ID="AtomicPlace">
  <rdfs:subClassOf rdf:resource="#Place"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#isSpatiallySubsumedBy"/>
      <owl:allValuesFrom
         rdf:resource="#CompoundPlace"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#spatiallySubsumes"/>
      <owl:cardinality>0</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AtomicPlaceInBuilding"/>
    <owl:Class
       rdf:about="#AtomicPlaceNotInBuilding"/>
  </owl:unionOf>
</owl:Class>
```

```
<owl:Class rdf:ID="CompoundPlace">
  <rdfs:subClassOf rdf:resource="#Place"/>
  <owl:disjointWith rdf:resource="#AtomicPlace"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#isSpatiallySubsumedBy"/>
      <owl:allValuesFrom
         rdf:resource="#CompoundPlace"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#spatiallySubsumes"/>
      <owl:allValuesFrom
         rdf:resource="#Place"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Campus"/>
    <owl:Class rdf:about="#Building"/>
  </owl:unionOf>
</owl:Class>
```

Figure 3: Partial definitions of the `AtomicPlace` and `CompoundPlace` classes.

The containment property in COBRA-ONT is represented by the `spatiallySubsumes` and `isSpatiallySubsumedBy` class properties. These two class properties are defined as the inverse property of each other (i.e., if X spatially subsumes Y, then Y is spatially subsumed by X).

For the `AtomicPlace` class and its subclasses, the cardinality of their `spatiallySubsumes` property is restricted to zero, and the range of their `isSpatiallySubsumedBy` property is `CompoundPlace`. The function of these constraints is to express the idea that all individuals of the type `AtomicPlace` do not spatially subsume other physical locations, and they can be spatially subsumed by individuals of the type `CompoundPlace`.

Like the `AtomicPlace` class, the `CompoundPlace` class is also defined with special constraints on its containment properties. For this class and its subclasses, the range of the `spatiallySubsumes` is `Place`, and the range of the `isSpatiallySubsumedBy` property is `CompoundPlace`. The function of these constraints is to express the idea that all individuals of the type `CompoundPlace` can spatially subsume other individuals of the type either `AtomicPlace` or `CompoundPlace`, and they can be spatially subsumed by other `CompoundPlace` individuals.

In our ontology, predefined subclasses of `AtomicPlace` are `Room`, `Hallway`, `Stairway`, `Restroom`, and `ParkingLot`, and predefined subclasses of `CompoundPlace` are `Campus` and `Building`. Notice that some of the`AtomicPlace` subclasses could have been modeled as subclasses of `CompoundPlace` (e.g., `Room` can be thought as a compound place that spatially subsumes four corners of a

room). The choice that we have made in categorizing these ontological concepts is purely based on the type of context-aware applications that we need to support in prototyping CoBrA. It may be necessary to re-organize the class hierarchy if the ontology is reused to support a different context-aware application.

To help to describe a place is hosting an event (e.g., a meeting), we define the `hasEvent` property. This property has domain `Place` and range `Event`. Instances of the `Event` class are associated with time intervals.

4.b.2 Ontologies about Agents

The top level agent class in COBRA-ONT is `Agent`. This class has two predefined subclasses, namely `Person` and `SoftwareAgent`. The former represents the class of all human agents, and the latter represents the class of all software agents. These two classes are defined to be disjoint. We have defined a number of properties for

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

<owl:ObjectProperty rdf:ID="fillsRole">
  <rdfs:domain rdf:resource="#Agent"/>
  <rdfs:range rdf:resource="#Role"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isFilledBy">
  <owl:inverseOf rdf:resource="#fillsRole"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="SpeakerRole">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#RoleInPresentation"/>
    <owl:Restriction>
      <owl:onProperty
rdf:resource="#intendsToPerform"/>
      <owl:hasValue
rdf:resource="#GivePPTPresentation"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="AudienceRole">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#RoleInPresentation"/>

    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#intendsToPerform"/>
      <owl:hasValue
         rdf:resource="#AttendPresentation"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

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

<owl:Class rdf:ID="ActionFoundInPresentation">
  <rdfs:subClassOf rdf:resource="#IntentionalAction"/>

  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#GivePPTPresentation"/>
    <owl:Thing rdf:about="#SetupPPTPresentation"/>
    <owl:Thing rdf:about="#AdjustLightIntensity"/>
    <owl:Thing rdf:about="#DistributeHangouts"/>
    <owl:Thing rdf:about="#AttendPresentation"/>
    <owl:Thing rdf:about="#AcquireHangouts"/>
    <owl:Thing rdf:about="#ExchangeContact"/>
  </owl:oneOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="intendsToPerform">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
               <owl:Class rdf:about="#Role"/>
               <owl:Class rdf:about="#Agent"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#IntentionalAction"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="desiresSomeoneToAchieve">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
               <owl:Class rdf:about="#Role"/>
               <owl:Class rdf:about="#Agent"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#IntentionalAction"/>
</owl:ObjectProperty>
```

Figure 4: Partial definitions of the classes related roles, intentions and desires

describing the profile of an agent (e.g., names, home pages, and email addresses). Each agent in our ontology can have associated roles in an event (e.g., during a presentation event, the role of a person is a speaker, and after the presentation event, the role of the same person changes to a meeting participant). The role of an agent is defined by the `fillsRole` property, which has range `Role`. For convenience, we predefined two subclasses of `Role`, `SpeakerRole` and `AudienceRole`. They represent different roles of a human agent in a meeting.

In our ontology, the role of an agent can be used to characterize the intention of the agent. This allows the system to reason about the possible actions that a user intends

to take after knowing the role of the user. To describe a user's intended action, we have defined the property `intendsToPerform` for the `Role` class. The range of this property is `IntentionalAction`.

Sometimes an agent may desire other agents to achieve certain objectives on its behalf. For example, a speaker may desire services to set up the presentation slides before the meeting starts. To define what actions an agent desires other agents to take, we define a property called `desiresSomeoneToAchieve`. The range of this property is `IntentionalAction`[3].

4.b.3 Ontologies about an Agent's Location Context

By location context, we mean a collection of dynamic knowledge that describes the location of an agent. The location property of an agent is represented by the property `locatedIn`. As the physical locations are categorized into `AtomicPlace` and `CompoundPlace`, it is possible to define the following context reasoning:

1. *No agent can be physically present in two different atomic places during the same time interval.*

2. *An agent can be physically present in two different compound places during the same time interval just in case one spatially subsumes the other.*

This type of reasoning is important because they can help the broker to detect inconsistent knowledge about the current location of an agent. For example, if two different sensor agents report a person is currently located in the Parking Lot A and is located in the Room 210, respectively, then based on the first rule, the broker can conclude the information about the person's location is inconsistent because both instances that represent the Parking Lot A and the Room 210 are type of the atomic place.

To describe an agent is physically present in an atomic or a compound place, from the `locatedIn` property we define two sub-properties called `locatedInAtomicPlace` and `locatedInCompoundPlace`. The former is defined with the range restricted to `AtomicPlace`, and the latter is defined with the range restricted to `CompoundPlace`. From these two properties, we define additional properties that further restricts the type of the physical place that an agent can have physical presence in. For example, `locatedInRoom`, `locatedInRestroom` and `locatedInParkingLot` are sub-properties of `locatedInAtomicPlace`; `locatedInCampus` and `locatedInBuiding` are sub-properties of `locatedInCompoundPlace`.

For agents that are located in different places, we can categorize them according to their location properties. For example, we define `PersonInBuilding` to represent a set of all people who are located in a building, and `SoftwareAgentInBuilding` to represent a set of all software agents that are located in a building. The complement of these classes are `PersonNotInBuilding` and `SoftwareAgentNotInBuilding`.

[3]The semantic of an action is not formal defined in v0.2 of the ontology. At present all action instances are assumed to be atomic actions.

4.b.4 Ontologies about an Agent's Activity Context

The activity context of an agent, similar to the location context, is a collection of dynamic knowledge that describes the events in which an agent participates. Events are assumed have schedules. In our ontology, the class `PresentationSchedule` represents the schedule of a presentation event. This class has associated properties that describe the start time, the end time, the presentation title, the presentation abstract, and the location of a presentation event. Additionally, in COBRA-ONT we also provided a set of constructs for describing the audiences and speakers of a presentation event. We assume in each presentation, there is at least one invited speaker and one or many audiences. To describe a presentation that has a speaker or an audience, one can use the property `invitedSpeaker` and `expectedAudience`. Both of these properties have domain `PresentationSchedule` and range `Person`.

To describe an event that is currently happening, we define a class called `PresentationEventHappeningNow`. The individuals of this class are assumed to have implicit association with the temporal predicate "now".

Sometimes it is useful to reason about the temporal property of the people and places that are associated a presentation event. For example, the broker might want to reason who is currently participating in a meeting, or what room is currently hosting a meeting. To support this type of reasoning, we defined the class `RoomHasPresentationEventHappeningNow` to represent the rooms that are currently hosting meetings, the class `SpeakerOfPresentationHappeningNow` to represent the speakers of the presentations that are currently happening, and the class `AudienceOfPresentationHappeningNow` to represent the audiences of the presentations that are currently happening.

5 An OWL Inference Engine in Flora-2

In order to support ontology reasoning in CoBrA, we have prototyped an OWL inference engine called F-OWL. This inference engine is implemented using Flora-2, an object-oriented knowledge base language and application development platform that translates a unified language of F-logic, HiLog, and Transaction Logic into the XSB deductive engine [25]. Key features of F-OWL include the ability to reason with the ontology model defined by the latest standard OWL language recommended by W3C, the ability to support knowledge consistency checking using axiomatic rules defined in Flora-2, and an open application programming interface (API) for Java application integrations.

The use of an object-oriented rule based language (i.e., Flora-2) in an advanced Prolog logic programming system (i.e., XSB) differentiates the implementation of F-OWL from other ontology inference engines such as JTP [11], RACER [14] and DAML-JessKB [17]. First, F-OWL exploits a special rule evaluation mechanism called tabling, provided by the underlying XSB system. This mechanism implements result caching in the backward chaining reasoning, which is beyond the capability of a traditional Prolog system. As ontology reasoning often involves repetitive evaluation of some closed-world domain knowledge, the tabling mechanism can help to avoid repetitive ontology

inference calculations, improving the overall system performance. Second, in contrast to the use of conventional logic languages in other ontology inference engines (e.g., KIF in JTP and CLIPS in DAMLJessKB), F-OWL adopts an object-oriented language Flora-2 that has a closer language constructs to the OWL language (e.g., both Flora-2 and OWL support the representations of classes, properties, restrictions, and instances). Third, building F-OWL on an advanced logic programming system creates opportunities for applications to be integrated into and interoperate with other intelligent systems (e.g., integrating an existing planning system to exploit knowledge inferred from an existing ontology model).

F-OWL is a rule-driven logic inference engine, which consists of the following four components: (i) assertions for the triple representation of the RDF and RDFS data models, (ii) assertions for the triple representation of the OWL data model, (iii) rules for reasoning with the RDF and RDFS data model, and (iv) rules for reasoning with the OWL data model. The latest version (v0.3)[4] of F-OWL supports the ontology reasoning over the RDFS and the OWL-Lite sub-language constructs.

To use F-OWL in context reasoning, the implementation of the context broker will provide additional rules to reason over the domain-specific knowledge. Rules of this type are the rules for detecting and resolving knowledge inconsistency and the rules for interpreting sensing inputs. In our prototype implementation, we have developed rules that reason about the location of a person in UMBC and the roles that are associated different participants in a scheduled meeting.

6 Future Work

An important next step of our work is to revise COBRA-ONT to use, if possible, or at least to map to, if feasible, the emerging consensus ontologies that are relevant to the development of smart spaces. These include the ontology for describing people on the Web (e.g., the Friends-Of-A-Friend Vocabulary Specification [2]), the ontology for describing time (e.g., the DAML-Time ontology) and space (e.g., the Relation Connection Calculus [21] and the DAML-Space ontology), and the ontology for describing talks (e.g., the ITTalks ontology [10]).

Modeling time is important in CoBrA. We currently have an implicit representation of time and temporal relations. In the next version, we plan on using the DAML-Time ontology, which is an ontology for expressing temporal aspects of the contents of web resources and for expressing time-related properties of web services (unpub. A DAML Ontology of Time. Nov. 2002). In DAML-Time, interval algebra is used to define temporal relationship axioms (after, before, inside, time-between, proper-interval, etc.) and representations for clock and calendar units (i.e., year, month, day of week, etc.).

We plan to use this ontology to model the temporal relations of different events in an intelligent meeting room. We will also develop rules that implement interval algebra to reason over the temporal relations of the described events. For example, using the `at-time(e,t)` and `inside(t,T)` predicates in the interval algebra, we can create rules to determine if a person is attending a meeting at a given time interval. In

[4] `http://umbc.edu/~hchen4/fowl`

this example, the lower case `e` represents an event instance, the lower case `t` represents a time instance and the upper case `T` represents a time interval. In an intelligent meeting room, sensors periodically reports the presence of a person to the broker and describe this information using the `at-time` predicate – e.g., at 1:05 PM, they report `at-time(located(harry,room201), t_instant("1:05PM"))`

Knowing there is a meeting scheduled in the Room 201 during the time interval 1:00PM-2:00PM, using the `inside(t,T)` axiom (see Figure 5), the broker concludes that Harry is located in the Room 201 during the meeting. Not knowing any evidence to the contrary, the broker may also conclude that Harry is attending the meeting.

$$\text{inside(t,T)} \Leftarrow \text{begins(t1,T)} \ \& \ \text{ends(t2,T)} \ \& \ \text{before(t1,t)} \ \& \ \text{before(t,t2)}$$

Figure 5: An instant is inside a proper interval if the beginning of the interval is before the instant, and the instant is before the end of the interval.

7 Conclusion

Ontologies are key requirements for building pervasive context-aware systems, in which independently developed sensors, devices, and agents are expected to share contextual knowledge and to provide relevant services and information to users based on their situational needs. We have described COBRA-ONT an ontology that we have developed for the Context Broker Architecture.

Our work shows that the newly emerged Web Ontology Language OWL is suitable for building a common knowledge representation for context-aware systems to share and reason with contextual knowledge. However, we also realize that a major shortcomings of our current design is in the inability to reuse other consensus ontologies. The disadvantages of building a complete ontology from the scratch are the following: (i) it potentially requires a larger amount of overhead in the ontology design and engineering, and (ii) it could decrease the interoperability between independently developed ontologies.

As a part of our long term research plan, we are prototyping an intelligent meeting room called EasyMeeting to demonstrate the feasibility of our Context Broker Architecture. Our goal is to deploy a pervasive context-aware meeting room in the newly constructed Information Technology and Engineering Building on the UMBC main campus.

References

[1] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *OWL Web Ontology Language Reference*, w3c candidate recommendation 18 august 2003 edition, August 2003.

[2] Dan Brickley and Libby Miller. *FOAF Vocabulary Specification*, revision 1.47 edition, Sept 2003.

[3] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven A. Shafer. Easyliving: Technologies for intelligent environments. In *Proceedings of Second International Symposium on Handheld and Ubiquitous Computing*, pages 12–29, 2000.

[4] 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.

[5] 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.

[6] Michael Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin. Meeting the computational needs of intelligent environments: The metaglue system. In *Proceedings of In 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99)*, Dublin, Ireland, 1999.

[7] Michael H. Coen. Building brains for rooms: Designing distributed software agents. In *Proceedings of Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 971–977, 1997.

[8] Michael H. Coen. Design principles for intelligent environments. In *Proceedings of AAAI/IAAI 1998*, pages 547–554, 1998.

[9] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deb McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *DAML+OIL Reference Description*, march 2001 edition, 2001.

[10] R. Scott Cost, Tim Finin, Anupam Joshi, Yun Peng, Charles Nicholas, Harry Chen, Lalana, Filip Perich, Youyong Zou, Sovrin Tolia, and Ian Soboroff. Ittalks: A case study in the semantic web and daml. In *Proceedings of the International Semantic Web Working Symposium*, July 2002.

[11] Richard Fikes, Jessica Jenkins, and Gleb Frank. Jtp: A system architecture and component library for hybrid reasoning. In *Proceedings of the Seventh World Multiconference on Systemics, Cybernetics, and Informatics*, July 2003.

[12] Tim Finin, Anupam Joshi, Lalana Kagal, Olga Ratsimore, Vlad Korolev, and Harry Chen. Information agents for mobile and embedded devices. *Lecture Notes in Computer Science*, 2182:264–??, 2001.

[13] The Foundations for Intelligent Physical Agents. *FIPA Abstract Architecture Specification*, sc00001l edition, December 2002.

[14] Volker Haarslev and Ralf Möller. Racer system description. In *Proceedings of the International Joint Conference on Automated Reasoning 2001*, 2001.

[15] Jeff Heflin. *Web Ontology Language (OWL) Use Cases and Requirements*, w3c candidate recommendation 18 august 2003 edition, 2003.

[16] Tim Kindberg and John Barton. A web-based nomadic computing system. *Computer Networks*, 35(4):443–456, 2001.

[17] Joe Kopena and William C. Regli. Damljesskb: A tool for reasoning with semantic web. *IEEE Intelligent Systems*, 18(3):74–77, May/June 2003.

[18] 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.

[19] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C, w3c recommendation 22 february 1999 edition, Feb 1999.

[20] Stephen Peters and Howie Shrobe. Using semantic networks for knowledge representation in an intelligent environment. In *1st Annual IEEE International Conference on Pervasive Computing and Proceedings of the 1st Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'03)*, March 2003.

[21] David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning*, 1992.

[22] Konstantinos Sagonas, Terrance Swift, David S. Warren, Juliana Freire, Prasad Rao, Baoqiu Cui, and Ernie Johnson. *The XSB Programmers' Manual*, version 2.6 edition, June 2003.

[23] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of CHI'99*, pages 434–441, 1999.

[24] 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.

[25] Guizhen Yang and Michael Kifer. *Flora-2: User's Manual*. Department of Computer Science, Stony Brook University, Stony Brook, release 0.92 edition, 2002.