The SOUPA Ontology for Pervasive Computing

Harry Chen, Tim Finin and Anupam Joshi

Abstract. This paper describes SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) and the use of this ontology in building the Context Broker Architecture (CoBrA). CoBrA is a new agent architecture for supporting pervasive context-aware systems in a smart space environment. The SOUPA ontology is expressed using the Web Ontology Language OWL and includes modular component vocabularies to represent intelligent agents with associated beliefs, desire, and intentions, time, space, events, user profiles, actions, and policies for security and privacy. Central to CoBrA is an intelligent broker agent that exploits ontologies to support knowledge sharing, context reasoning, and user privacy protection. We also describe two prototype systems that we have developed to demonstrate the feasibility and the use of CoBrA.

Keywords. Semantic web ontology, pervasive computing, smart spaces, agents.

1. Introduction

Pervasive computing is a natural extension of the existing computing paradigm. In the pervasive computing vision, software agents, services, and devices 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 [17]. An important next step for pervasive computing is the integration of intelligent agents that employ knowledge and reasoning to understand the local context and share this information in support of intelligent applications and interfaces. We describe a new architecture called the Context Broker Architecture (CoBrA) for supporting context-aware systems in smart spaces (e.g., intelligent meeting rooms, smart homes, and smart vehicles).

A key difference between CoBrA and the existing pervasive computing systems [15, 11, 37, 39, 26, 23] is in the use of ontology [7]. Computing entities in

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

CoBrA can share context knowledge using the CoBrA ontology, which extends the SOUPA ontology (Standard Ontology for Ubiquitous and Pervasive Applications) [10]. These ontologies are expressed in the Web Ontology Language OWL [29]. Central to CoBrA is an intelligent broker agent called *Context Broker*. Using the CoBrA ontology and the associated logic inference engines, the Context Broker can reason about the local context of a smart space, and detect and resolve inconsistent context knowledge acquired from disparate sensors and agents. Human users in the CoBrA system can use an OWL representation of the Rei policy language [22] to define privacy policies to control the sharing of their private information in a pervasive context-aware environment.

Context-aware systems are computer systems that can provide relevant services and information to users by exploiting context [6]. 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 that it contains. Context may also include system capabilities, services offered and sought, the activities and tasks in which people and computing entities are engaged, and their situational roles, beliefs, and intentions [7].

The design of CoBrA addresses the following research issues in building pervasive context-aware systems: (i) *context modeling* (i.e., how to represent and store contextual information), (ii) *context reasoning* (i.e., how to interpret context based on the information acquired from the physical environment; how to detect and resolve inconsistent context knowledge due to inaccurate sensing), (iii) *knowledge sharing* (i.e., how to help independently developed computing entities to share context knowledge and interoperate), and (iv) *user privacy protection* (i.e., how to protect the privacy of users by restricting the sharing of contextual information acquired by the hidden sensors or agents).

The rest of this paper is organized as follows: In the next section, we describe the problems in existing pervasive computing systems and our motivation to use ontologies in CoBrA. In Section 3, we describe the SOUPA ontology and how it can be used to represent various types of contextual information. In Section 4, we present the design of CoBrA. In Section 5, we describe two different prototype systems that implement CoBrA. One is a prototype for supporting context-aware services in a smart meeting room system called EasyMeeting, and the other is a toolkit for building stand-alone demonstrations of the Context Broker. Future work and conclusions are given in Section 6 and Section 7, respectively.

2. Problems in the Existing Pervasive Computing Systems

A number of pervasive computing prototype systems have been designed and implemented. Contributions to the field have been made in various aspects of pervasive computing. Dey [15] developed a middle-aware framework to facilitate context

acquisition, Coen et al. [11] defined new extensible programming libraries for building intelligent room agents, and several groups [37, 39] have created badge-size tracking devices for determining people's location in an indoor environment.

Major shortcomings of these systems are that they are weak in supporting knowledge sharing and reasoning and lack adequate mechanisms to control how information about individuals is used and shared with others. In Dey's Context Toolkit framework [15], Schilit's context-aware architecture [37], and the Active Badge system [39], context knowledge is embedded in programming objects (e.g., Java classes) that are often inadequate for supporting knowledge sharing and data fusion operations. The designs of these systems also make strong assumptions about the accuracy of the information acquired from the hardware sensors. In an open and dynamic environment, such assumptions can lead to system implementations that cannot cope with the frequently occurred inconsistent context knowledge. In the Intelligent Room system [12] and the Cooltown architecture [26], information about a user can be freely shared by all computing entities in the environment. As physical environments are populated with ambient sensors, users may be unaware of the use and the sharing of their private information, which can create great concerns for privacy.

The design of CoBrA is aimed to address these issues using ontologies. 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 system [33], (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" [19].

3. The SOUPA Ontology

The SOUPA project began in November 2003 and is part of an ongoing effort of the Semantic Web in UbiComp Special Interest Group[1], an international group of researchers from academia and industry that is using the OWL language for pervasive computing applications and defining ontology-driven use cases demonstrating aspects of the ubiquitous computing vision. The SOUPA ontology is expressed using the OWL language and includes modular component vocabularies to represent intelligent agents with associated beliefs, desires, and intentions, time, space, events, user profiles, actions, and policies for security and privacy.

The goal of the project is to define ontologies for supporting pervasive computing applications. The design of SOUPA is driven by a set of use cases. While the SOUPA vocabularies overlap with the vocabularies of some existing ontologies, the merits of SOUPA is in providing pervasive computing developers a shared

[1] `http://pervasive.semanticweb.org`

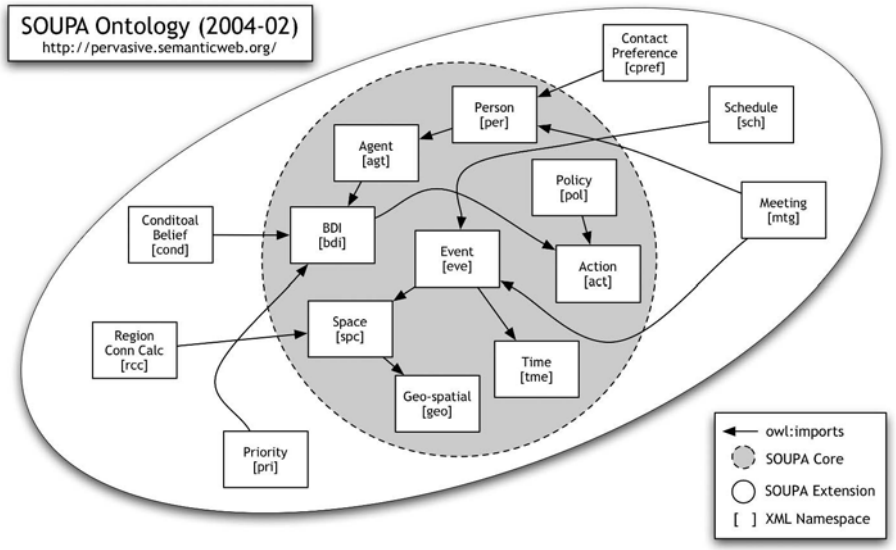

FIGURE 1. SOUPA consists of two sets of ontology documents: SOUPA Core and SOUPA Extension. The OWL `owl:imports` construct is used to enable a modular design of the ontology. Different domain vocabularies are grouped under different XML namespaces.

ontology that combines many useful vocabularies from different consensus ontologies. By providing a shared ontology, SOUPA can help developers inexperienced in knowledge representation to quickly begin building ontology-driven applications without needing to define ontologies from scratch and to be more focused on the functionalities of actual system implementations.

SOUPA consists of two distinctive but related set of ontologies: *SOUPA Core* and *SOUPA Extension*. The set of the SOUPA Core ontologies attempts to define generic vocabularies that are universal for building pervasive computing applications. The set of SOUPA Extension ontologies, extended from the core ontologies, define additional vocabularies for supporting specific types of applications and provide examples for defining new ontology extensions.

Note that the structure of the ontology merely suggests certain vocabularies are more general than the others in supporting pervasive computing applications, and there is no inherent computational complexity difference in adopting either set of the ontologies. The complete set of SOUPA ontologies is available at `http://pervasive.semanticweb.org`.

3.1. The Web Ontology Language OWL

The OWL language is a Semantic Web language for use by computer applications that need to process the content of information instead of just presenting information to humans [29]. This language is developed in part of the Semantic Web initiatives sponsored by the World Wide Web Consortium (W3C).

The current human-centered web is largely encoded in HTML, which focuses largely on how text and images would be rendered for human viewing. Over the past few years we have seen a rapid increase in the use of XML as an alternative encoding, one that is intended primarily for machine processing. The machine which process XML documents can be the end consumers of the information, or they can be used to transform the information into a form appropriate for humans to understand (e.g., as HTML, graphics, and synthesized speech). As a representation language, XML provides essentially a mechanism to declare and use simple data structures, and thus it leaves much to be desired as a language for expressing complex knowledge. Enhancements to the basic XML, such as XML Schemas, address some of the shortcomings, but still do not result in an adequate language for representing and reasoning about the kind of knowledge essential to realizing the Semantic Web vision.

OWL is a knowledge representation language for defining and instantiating ontologies. An ontology is a formal explicit description of concepts in a domain of discourse (or classes), properties of each class describing various features and attributes of the class, and restrictions on properties [30].

The normative OWL exchange syntax is RDF/XML. Ontologies expressed in OWL are usually placed on web servers as web documents, which can be referenced by other ontologies and downloaded by applications that use ontologies. In this paper, we refer to these web documents as *ontology documents*.

3.2. Related Ontologies

Part of the SOUPA vocabularies are adopted from a number of different consensus ontologies. The strategy for developing SOUPA is to borrow terms from these ontologies but not to import them directly. Although the semantics for importing ontologies is well defined [1], by choosing not to use this approach we can effectively limit the overhead in requiring reasoning engines to import ontologies that may be irrelevant to pervasive computing applications. However, in order to allow better interoperability between the SOUPA applications and other ontology applications, many borrowed terms in SOUPA are mapped to the foreign ontology terms using the standard OWL ontology mapping constructs (e.g., `owl:equivalentClass` and `owl:equivalentProperty`).

The ontologies that are referenced by SOUPA include the Friend-Of-A-Friend ontology (FOAF) [3, 35], DAML-Time and the entry sub-ontology of time [20, 31], the spatial ontologies in OpenCyc [27], Regional Connection Calculus (RCC) [36], COBRA-ONT [7], MoGATU BDI ontology [32], and the Rei policy ontology [21]. In the rest of this section, we describe the key features of these related ontologies and point out their relevance to pervasive computing applications.

FOAF. This ontology allows the expression of personal information and relationships, and is a useful building block for creating information systems that support online communities [16]. Pervasive computing applications can use FOAF ontologies to express and reason about a person's contact profile and social connections to other people in their close vicinity.

DAML-Time & the Entry Sub-ontology of Time. The vocabularies of these ontologies are designed for expressing temporal concepts and properties common to any formalization of time. Pervasive computing applications can use these ontologies to share a common representation of time and to reason about the temporal orders of different events.

OpenCyc Spatial Ontologies & RCC. The OpenCyc spatial ontologies define a comprehensive set of vocabularies for symbolic representation of space. The ontology of RCC consists of vocabularies for expressing spatial relations for qualitative spatial reasoning. In pervasive computing applications, these ontologies can be exploited for describing and reasoning about location and location context [7].

COBRA-ONT & MoGATU BDI Ontology. Both COBRA-ONT and MoGATU BDI ontology are aimed for supporting knowledge representation and ontology reasoning in pervasive computing environment. While the design of COBRA-ONT focuses on modeling contexts in smart meeting rooms [7], the design of MoGATU BDI ontology focuses on modeling the belief, desire, and intention of human users and software agents [32].

Rei Policy Ontology. The Rei policy language defines a set of deontic concepts (i.e., rights, prohibitions, obligations, and dispensations) for specifying and reasoning about security access control rules. In a pervasive computing environment, users can use this policy ontology to specify high-level rules for granting and revoking the access rights to and from different services [24].

3.3. SOUPA Core

The SOUPA core ontology consists of nine ontology documents. Together these ontology documents define vocabularies for describing person contact information, beliefs, desires, and intentions of an agent, actions, policies, time, space, and events.

Person. This ontology defines typical vocabularies for describing the contact information and the profile of a person. The OWL class `per:Person` is defined to represent a set of all people in the SOUPA domain, and is equivalent to the `foaf:Person` class in the FOAF ontology (i.e., the `owl:equivalentClass` property holds between the `per:Person` and `foaf:Person` class). An individual of the class can be described by a set of properties, which include basic profile information (name, gender, age, birth date, etc.), the contact information (email, mailing address, homepage, phone numbers, instant messaging chat ID, etc.), and social and professional profile (people that a person is friend of, organizations that a person belongs to). In addition, all property vocabularies that are applicable to

describe a person in the FOAF ontology can also be used to describe an individual of the `per:Person` class. This is because all individuals of the `per:Person` class are also individuals of the `foaf:Person` class. Figure 2 shows a partial ontology description of the person Harry Chen.

Agent, Action & BDI. Sometimes when building intelligent pervasive computing systems, it is useful to model computing entities as *agents* [40]. In SOUPA, agents are defined with a strong notion of agency [40], which is characterized by a set of *mentalistic* notions such as knowledge, belief, intention, and obligation. In this ontology, both computational entities and human users can be modeled as agents.

When the goals, plans, desires, and beliefs of different agents are explicitly represented in the ontologies, this information can help independently developed agents to share a common understanding of their "mental" states, helping them to cooperate and collaborate. The explicitly represented human user's mental states can help computing agents to reason about the specific needs of the users in a pervasive environment.

Three ontology documents are related to this ontology: `agent`, `bdi`, and `action`. The `agt:Agent` class represents a set of all agents in the SOUPA domain and is associated with three properties that can be used to characterize an agent's "mental" state: `agt:believes`, `agt:desires`, and `agt:intends`. The respective range values of these properties are the `bdi:Fact`, `bdi:Desire`, and `bdi:Intention` classes. The goals of an agent are considered to be a special type of desire, which is expressed by defining the `agt:hasGoal` property as a sub-property of the `agt:desires` property.

The `bdi:Fact` class is a subclass of the `rdf:Statement` class, which represents a set of reified RDF statements [2]. A reified RDF statement consists of the `rdf:subject`, `rdf:object`, and `rdf:predicate` properties.

The `bdi:Desire` class defines a set of world states that agents desire to bring about. Every instance of this class can be characterized by the property `bdi:endState`. The range restriction of this property is unspecified in the `bdi` ontology document. Application developers are responsible for defining the representation of different world states. Some suggested representations are (i) symbolic names, e.g., a set of pre-defined RDF resource URI and (ii) meta-representation, e.g., each world state description is a set of reified RDF statements.

The `bdi:Intention` class represents a set of plans that agents intend to execute. Plans are defined in terms of actions, pre-conditions, and effects. The `action` ontology document defines the `act:Action` class with associated properties `act:preCondition` and `act:effect`. The representation of pre-conditions and effects are unspecified in this ontology, and it is left to be defined by the application ontologies.

Sometimes it may be useful to describe whether or not different desires of an agent are in conflict of each other, and whether or not certain desires are achievable. The cause of desire conflicts may be due to inconsistent beliefs in the knowledge base or conflicting user preferences or systems policies. The cause

```
<!DOCTYPE rdf:RDF [
   ...
   <!ENTITY foaf   "http://xmlns.com/foaf/0.1/#">
   <!ENTITY per    "http://pervasive.semanticweb.org/dev/person#">
   ]>

<rdf:RDF ... >

<per:Person>
  <per:firstName rdf:datatype="&xsd;string">Harry</per:firstName>
  <per:lastName rdf:datatype="&xsd;string">Chen</per:lastName>

  <per:gender rdf:resource="&per;Male"/>

  <per:birthDate rdf:datatype="&xsd;date">1976-12-26</per:birthDate>

  <per:homepage rdf:resource="http://umbc.edu/people/hchen4"/>

  <foaf:weblog rdf:resource="http://umbc.edu/people/hchen4"/>

  <per:hasSchoolContact rdf:resource="#SchoolContact"/>
  <per:hasHomeContact rdf:resource="#HomeContact"/>

  <foaf:workplaceHomepage rdf:resource="http://ebiquity.umbc.edu"/>
  <foaf:workplaceHomepage rdf:resource="http://www.umbc.edu"/>
  <foaf:workplaceHomepage rdf:resource="http://www.cs.umbc.edu"/>
</per:Person>

<per:ContactProfile rdf:ID="SchoolContact">
  <per:address rdf:datatype="&xsd;string">
    Dept. of CSEE, UMBC, 1000 Hilltop Circle, Baltimore, MD 21250, USA
   </per:address>
   <per:phone rdf:datatype="&xsd;string">+1-410-455-8648</per:phone>
   <per:email rdf:resource="harry.chen@umbc.edu"/>
   <per:im rdf:resource="aim:goim?screenname=hc1379"/>
</per:ContactProfile>

<per:Email rdf:about="harry.chen@umbc.edu"/>

<per:Homepage rdf:about="http://www.aim.com"/>

<per:ChatID rdf:about="aim:goim?screenname=hc1379">
  <per:providedBy rdf:resource="http://www.aim.com"/>
</per:ChatID>

<per:ContactProfile rdf:ID="HomeContact">
  ...
</per:ContactProfile>

<foaf:knows>
  <foaf:Person>
     <foaf:name>Tim Finin</foaf:name>
     <foaf:mbox_sha1sum>49953f47b9c33484a753eaf14102af56c0148d37</foaf:mbox_sha1sum>
  </foaf:Person>
</foaf:knows>
</rdf:RDF>
```

FIGURE 2. A partial ontology description of the person Harry Chen. Vocabularies from both the SOUPA ontology and the FOAF ontology can be used to describe a person.

```
<!DOCTYPE rdf:RDF [
   ...
   <!ENTITY jade   "http://pervasive.semanticweb.org/dev/fipa-jade#>
   <!ENTITY pol    "http://pervasive.semanticweb.org/dev/policy#">
   ]>
<rdf:RDF>
<pol:Policy>

  <pol:creator>
    <per:Person>
      <foaf:name>Tim Finin</foaf:name>
    </per:Person>
  </pol:creator>
  <pol:enforcer>
    <agt:Agent>
      <jade:name rdf:datatype="&xsd;string">Context Broker in ITE328</jade:name>
      <jade:agentID rdf:datatype="&xsd;string">ctb@cobra1.cs.umbc.edu:1099/JADE</jade:agentID>
    </agt:Agent>
  </pol:enforcer>

  <pol:rule rdf:resource="#r1"/>
  <pol:rule rdf:resource="#r2"/>
</pol:Policy>

<pol:Prohibition rdf:ID="r1">
  ...
</pol:Prohibition>

<pol:Right rdf:ID="r2">
  ...
</pol:Right>

</rdf:RDF>
```

FIGURE 3. An instance of the `pol:Policy` class is the entry point to a SOUPA policy definition. In this example, the policy is created by a person with `foaf:name` "Tim Finin", and is enforced by the Context Broker in the Room ITE328. The definition of the two associated policy rules (i.e., `r1` and `r2`) is shown in the next figure.

of unachievable desires may be due to the change of situational conditions. In the `bdi` ontology document, different subclasses of the `bdi:Desire` class, namely `bdi:ConflictingDesire`, `bdi:NonConflictingDesire`, `bdi:AchievableDesire`, and `bdi:UnachievableDesire`, are defined for classifying different types of agent desires.

Policy. Security and privacy are two growing concerns in developing and deploying pervasive computing systems [4, 25, 18]. Policy is an emerging technique for controlling and adjusting the low-level system behaviors by specifying high-level rules [14].

Part of the SOUPA policy ontology adopts the vocabularies of the Rei policy language [21]. In SOUPA, a policy is a set of rules. Rules are defined by a *policy creator* (e.g., a user or an agent), and the rules are to be enforced by one or more *policy enforcer* (e.g., a security authority or a privacy protection agent). The

```
<pol:Prohibition rdf:ID="r1">
  <!-- Any owl:Thing that is an audience of the defined Ebiquity Group meeting
       is prohibited from changing presentation slides. -->
  <pol:actor>
    <!-- a special class for expressing "For all X"-->
    <pol:Variable/>
  </pol:actor>

  <pol:action rdf:resource="ChangePresentationSlides"/>

  <pol:constraintMemebershipClass>
    <owl:Class>
      <rdf:subClassOf rdf:resource="&mtg;Audience"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&mtg;ofMeeting"/>
          <owl:hasValue rdf:resource="http://ebiquity.umbc.edu/v2.1/event/html/id/15/"/>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
  </pol:constraintMemebershipClass>
</pol:Prohibition>

<pol:Right rdf:ID="r2">
  <!-- Person Harry Chen has the right to adjust room lighting if he is
       a speaker of the defined Ebiquity Group meeting -->

  <pol:actor>
    <per:Person>
      <foaf:name>Harry Chen</foaf:name>
    </per:Person>
  </pol:actor>

  <pol:action rdf:resource="#AdjustRoomLighting"/>

  <pol:constraintMemebershipClass>
    <owl:Class>
      <rdf:subClassOf rdf:resource="&mtg;Speaker"/>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&mtg;ofMeeting"/>
          <owl:hasValue rdf:resource="http://ebiquity.umbc.edu/v2.1/event/html/id/15/"/>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
  </pol:constraintMemebershipClass>
</pol:Right>
```

FIGURE 4. The `pol:Prohibition` and `pol:Right` are subclasses of the `pol:DenoticObject` class. Instances of these classes represent individual SOUPA policy rules.

definition of each rule gives specific enforcement instructions to the policy enforcer over a set of actions. For example, an action may be adjusting the lighting in a room, changing presentation slides on a projector device, or printing documents to a nearby printer.

Every enforcement instruction given to the policy enforcer falls under one of the following four categories: (i) the enforcer should permit the agents of certain class to perform the specified action, (ii) the enforcer should prohibit the agents

of certain class to perform the specified action, (iii) the enforcer should assign the agents of certain class to be responsible for performing the specified action, and (iv) the enforcer should waive the agents of certain class to be responsible for performing the specified action.

The entry point to the SOUPA policy ontology is the `pol:Policy` class. An individual of this class represents a policy document. The property `pol:rule` relates a policy rule instance to a policy document. Each policy document can have zero or more defined policy rules. Figure 3 shows a partial description of a SOUPA policy.

Policy rules are typically defined as individuals of one of the four rule classes: (i) `pol:Right`, (ii) `pol:Prohibition`, (iii) `pol:Obligation`, and (iv) `pol:Dispensation`. The semantics of these four class definitions correspond to the four enforcement instructions that are described above. These four classes are subclasses of the `pol:DenoticObject` class, and the set of individual members of each class disjoints with each other.

The `pol:DeonticObject` class has three defined properties: `pol:action`, `pol:actor`, and `pol:constraintMemebershipClass`. The `pol:action` property relates a policy rule to a specific action that it applies to, which must be type of `act:Action`. The `pol:actor` property defines a named agent who may be the actor of the defined action. By default, an actor does not have the right to perform an action unless it also satisfies the membership class constraint defined by the `pol:constraintMembershipClass` property.

The range of the `pol:constraintMemebershipClass` property is `owl:Class`. The purpose for this construct is to define a template class to match the class membership types of a given actor. An actor belongs to the constraint membership class if it is `rdf:type` of the defined class. In which case, the enforcement instruction given by the rule applies to the actor. Figure 4 shows examples of two SOUPA policy rules.

Time. SOUPA defines a set of ontologies for expressing time and temporal relations. They can be used to describe the temporal properties of different events that occur in the physical world.

Part of the SOUPA ontology adopts the vocabularies of the DAML-time ontologies and the entry sub-ontology of time. The basic representation of time consists of the `tme:TimeInstant` and `tme:TimeInterval` classes. All individual members of these two classes are also members of the `tme:TemporalEntity` class, which is an OWL class that is defined by taking the union of the `tme:TimeInstant` and `tme:TimeInterval` classes. The set of all temporal things that are divided into two disjoint classes: `tme:InstantThing`, things with temporal descriptions that are type of time instant, and `tme:IntervalThing`, things with temporal descriptions that are type of time interval. The union of these two classes forms the `tme:TemporalThing` class.

In order to associate temporal things with date/time values (i.e., their temporal descriptions), the `tme:at` property is defined to associate an instance of the

```
<tme:TimeInterval>
  <tme:from>
    <tme:TimeInstant>
      <tme:at rdf:datatype="xsd;dateTime">2004-02-01T12:01:01</tme:at>
    </tme:TimeInstant>
  </tme:from>
  <tme:to>
    <tme:TimeInstant>
      <tme:at rdf:datatype="xsd;dateTime">2004-02-11T13:41:21</tme:at>
    </tme:TimeInstant>
  </tme:to>
</tme:TimeInterval>
```

FIGURE 5. A representation of a time interval using the SOUPA time ontology. The beginning and the ending of a time interval are defined by the `tme:from` and `tme:to` properties, respectively.

`tme:InstantThing` with an XML `xsd:dateTime` datatype value (e.g., 2004-12-25T12:32:12), and the `tme:from` and `tme:to` properties are defined to associate an instance of the `IntervalThing` with two different `tme:TimeInstant` individuals. Figure 5 shows the representation of a time interval with the associated temporal description.

For describing the order relations between two different time instants, the ontology defines the following properties: `tme:before`, `tme:after`, `tme:beforeOrAt`, `tme:afterOrAt`, and `tme:sameTimeAs`. Both `tme:before` and `tme:after` properties are defined of type `owl:TransitiveProperty`. The `tme:sameTimeAs` property expresses that two different time instants are associated with equivalent date/time values and is defined of type `owl:SymmetricProperty`.

For describing the order relations between two different temporal things (i.e., time instants and time intervals), the ontology defines the following properties: `tme:startsSoonerThan`, `tme:startsLaterThan`, `tme:startsSameTimeAs`, `tme:endsSoonerThan`, `tme:endsLaterThan`, `tme:endsSameTimeAs`, `tme:startsAfterEndOf`, and `tme:endsBeforeStartOf`. The first three properties respectively express that for any two given temporal things A and B, the starting time of A is before the starting time of B, the starting time of A is after the starting time of B, and the starting time of A is the same as the starting time of B. The next three properties respectively express that for any two given temporal things A and B, the ending time of A is before the ending time of B, the ending time of A is after the ending time of B, and the ending time of A is the same as the ending time of B. The `tme:startsAfterEndOf` property expresses that the beginning of one temporal thing is after the ending of another temporal thing, and the `tme:endsBeforeStartOf` property expresses the inverse of this property.

Space. This ontology is designed to support reasoning about the spatial relations between various types of geographical regions, mapping from the geo-spatial coordinates to the symbolic representation of space and *vice versa*, and the representation of geographical measurements of space. Part of this ontology vocabularies are adopted from the spatial ontology in OpenCyc and the OpenGIS vocabularies [13]. Two ontology documents are related to this ontology: `space` and `geo-measurement`. The first ontology document defines a symbolic representation of space and spatial relations, and the second document defines typical geo-spatial vocabularies (e.g., longitude, latitude, altitude, distance, and surface area).

In the symbolic representation model, the `spc:SpatialThing` class represents a set of all things that have spatial extensions in the SOUPA domain. All spatial things that are typically found in maps or construction blueprints are called `spc:GeographicalSpace`. This class is defined as the union of the `spc:GeographicalRegion`, `spc:FixedStructure`, and `spc:SpaceInAFixedStructure` classes.

An individual member of the `spc:GeographicalRegion` class typically represents a geographical region that is controlled by some political body (e.g., the country USA is controlled by the US government). This relation is expressed by the `spc:controls` property, the domain of which is `spc:GeopoliticalEntity` and the range of which is `spc:GeographicalRegion`. Knowing which political entity controls a particular geographical region, a pervasive computing system can choose to apply the appropriate policies defined by the political entity to guide its behavior. For example, a system may apply different sets of privacy protection schemes based on the policies defined by the local political entities.

To support spatial containment reasoning, individual members of the `spc:GeographicalSpace` class can relate to each other through the `spc:spatiallySubsumes` and `spc:spatiallySubsumedBy` properties. For example, a country region may spatially subsume a state region, a state region may spatially subsume a building, and a building may spatially subsume a room. Knowing the room in which a device is located, we can infer the building, the state and the country that spatially subsumes the room.

In the geo-spatial representation model, the individual members of the `spc:SpatialThing` class are described by location coordinates (i.e., longitude, latitude, and altitude). This relation is expressed by the `spc:hasCoordinates` property, the range of which is the `geo:LocationCoordinates` class. In this model, multiple location coordinates can be mapped to a single geographical region (e.g., a university campus typically covers multiple location coordinates.). This relation is useful for defining spatial mapping between different geographical locations and GPS coordinates. This information can enable a GPS-enabled device to query the symbolic representation of its present location for a given set of longitude, latitude, and altitude.

Event. Events are event activities that have both spatial and temporal extensions. An event ontology can be used to describe the occurrence of different activities,

```
<owl:Class rdf:ID="DetectedBluetoothDev">
  <rdfs:subClassOf rdf:resource="&eve;TemporalSpatialEvent"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="foundDevice">
  <rdfs:domain rdf:resource="#DetectedBluetoothDev"/>
</owl:ObjectProperty>

<DetectedBluetoothDev>
  <spc:hasCoordinates>
    <geo:LocationCoordinates>
      <geo:longitude rdf:datatype...>-76.7113</geo:longitude>
      <geom:latitude rdf:datatype...>39.2524</geom:latitude>
    </geo:LocationCoordinates>
  </spc:hasCoordinates>

  <foundDevice rdf:resource="url-x-some-device"/>
  <tme:at>
    <tme:TimeInstant>
      <tme:at rdf:datatype="xsd;dateTime">2004-02-01T12:01:01</tme:at>
    </tme:TimeInstant>
  </tme:at>
<DetectedBluetoothDev>
```

FIGURE 6. An example shows the representation of a sensing event using the SOUPA space, time and event ontology. In this example, the Bluetooth network interface of a device has been detected at the time instant `2004-02-01T12:01:01` at a location with the GPS coordinates (-76.7113/39.2524).

schedules, and sensing events. In the `event` ontology document, the `eve:Event` class represents a set of all events in the domain. However, the definition of this class is silent about its temporal and spatial properties.

The `eve:SpatialTemporalThing` class represents a set of things that have both spatial and temporal extensions, and it is defined as the intersection of the `tme:TemporalThing` and `spc:SpatialThing` classes. To specifically describe events that have both temporal and spatial extensions, `eve:SpatialTemporal-Event` class is defined as the intersection of the `eve:SpatialTemporalThing` and `eve:Event` classes.

Figure 6 shows how the ontology can be used to describe an event in which a Bluetooth device has been detected on 2004-02-01 at 12:01:01 UTC, and the event occurs at a location that is described by longitude -76.7113 and latitude 39.2524.

3.4. SOUPA Extension

The SOUPA Extension ontologies are defined with two purposes: (i) to define an extended set of vocabularies for supporting specific types of pervasive application domains, and (ii) to demonstrate how to define new ontologies by extending the SOUPA Core ontologies. At present, the SOUPA Extension consists of experimental ontologies for supporting pervasive context-aware applications in smart spaces and peer-to-peer data management in a pervasive computing environment.

Priority. By default the BDI ontology is silent about the priority relation among the set of desires and intentions of an agent. The priority ontology defines additional vocabularies for assigning priority values to an agent's desires and intended actions. At times when there are conflicts between different desires or actions, priority values can be used to set the precedence.

Conditional & Unconditional Belief. This ontology defines the vocabularies for describing conditional beliefs. A conditional belief statement can be attributed by temporal values, accuracy values, or locally defined conditions. Statements defined with conditional attributes will be believed to be true if the associated time stamp is valid, the accuracy value is above a pre-defined threshold, and all the locally defined conditions are satisfied. Otherwise, the statements will be believed to be false.

Contact Preference. This ontology defines the vocabularies for describing a user's contact preference, which is a set of rules that specify how the user likes to be contacted by the system under different situational conditions (i.e., in meeting, out of town, on the weekends). For example, a user may specify the system to contact her on a cellphone when she is out of town, and to contact her using only SMS when she is in a meeting.

Meeting & Schedule. These two ontologies define the vocabularies for describing a meeting event, schedules, and the associated attendees. They can help smart meeting systems to represent and reason about the context of a meeting (e.g., are all scheduled attendees located in the meeting room? What is the end time of this meeting?)

4. The Context Broker Architecture

CoBrA is a broker-centric agent architecture for supporting context-aware systems in smart spaces [8]. Central to the architecture is the presence of a Context Broker, an intelligent agent that runs on a resource-rich stationary computer in the space. The responsibility of the Context Broker is to (i) provide a centralized model of context that can be shared by all devices, services, and agents in the space, (ii) acquire contextual information from sources that are unreachable by the resource-limited devices, (iii) reason about contextual information that cannot be directly acquired from the sensors (e.g., intentions, roles, temporal and spatial relations), (iv) detect and resolve inconsistent knowledge that is stored in the shared model of context, and (v) protect user privacy by enforcing policies that the users have defined to control the sharing and the use of their contextual information.

Our centralized design of the context broker is motivated by the need to support small devices that have relatively limited resources available for context acquisition and reasoning. With the presence of a broker, small devices such as cellphones, PDA and watches can offload their burdens of managing context knowledge onto a resource rich context broker, including reasoning with context, detecting

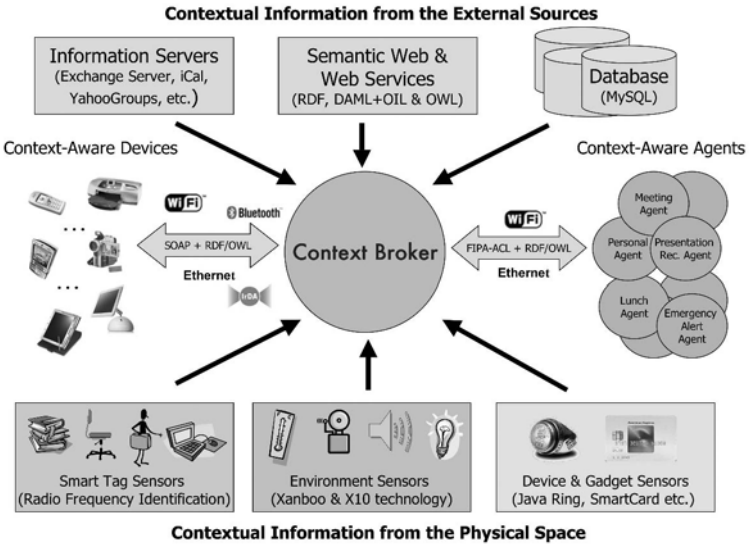

FIGURE 7. A Context Broker acquires contextual information from heterogeneous sources and fuses into a coherent model that is then shared with computing entities in the space.

and resolving inconsistent context knowledge. Furthermore, in an open and dynamic environment, users may desire that their personal contextual information be kept private. A centralized management of context knowledge makes it easy to implement privacy protection and information security.

The design of a Context Broker consists of the following four parts:

1. **CoBrA Ontology (COBRA-ONT):** A set of ontology documents that define the vocabularies for representing contextual information and for supporting the context reasoning. This ontology extends the SOUPA ontology and introduces additional domain specific vocabularies. COBRA-ONT v0.5[2] is the latest version of this ontology, which defines an ontology of the UMBC eBiquity Research Group meetings and a spatial ontology that describes the geographical location of UMBC.
2. **Context Knowledge Base:** This knowledge base stores the contextual information that is acquired from the physical environment and knowledge that is inferred by the Context Reasoning Engine. In our prototype implementation [9], this knowledge base is stored in a relational database[3] using the Persistent Ontology Model API of the Jena 2 semantic web framework [5]. All knowledge in this knowledge base is expressed as RDF triples.

[2]COBRA-ONT is available at `http://cobra.umbc.edu`.
[3]The current implementation uses the MySQL system.

3. **Context Reasoning Engine:** It is a rule-based component that provides logic inference support for interpreting context and for detecting knowledge inconsistency. This engine has a two-tier design: Tier-1 and Tier-2. A key difference between the reasoners in Tier-1 and Tier-2 is in the type of inferences that they support. While Tier-1 only supports ontology inferences using either the built-in or the customized ontology axioms, Tier-2 supports domain heuristics inferences using an external logic inference engine. We have prototyped two reasoners – one in Prolog [8] and the other one in Jess [9].
4. **Privacy Protection Module:** This module is responsible for analyzing user defined policy rules and helps the Context Broker to decide whether context knowledge about a user can be shared with a particular agent in the system. In our prototype design, this module reads in user privacy policies that are expressed in the SOUPA policy ontology. Before the Context Broker shares the context knowledge about a user, the Context Broker calls this module to check whether the receiving agent is permitted to acquire this information.

5. CoBrA Applications

To demonstrate the use and the feasibility of CoBrA for supporting pervasive context-aware systems, we have developed two prototype systems. Both prototypes exploit the SOUPA and COBRA-ONT ontology to support context modeling, context reasoning, and knowledge sharing. The first prototype system, called EasyMeeting, is a smart meeting room system that is aimed to facilitate typical user activities in an everyday meeting. The second prototype system is a toolkit for building demonstrations of the CoBrA system without needing to set up a complete pervasive computing infrastructure.

5.1. The EasyMeeting System

EasyMeeting is an extension to Vigil [38], a third generation pervasive computing infrastructure developed at UMBC. The goal of developing EasyMeeting is to create a smart meeting room that can facilitate typical user activities in an everyday meeting. This includes setting up presentations, allowing users to control services via speech, and adjusting lighting and background music in a room based the state of the meeting.

In EasyMeeting, the role of a Context Broker is to provide a shared model of context for all agents and services. In particular, it is responsible for acquiring and maintaining consistent knowledge about (i) the location of meeting participants, (ii) the event schedule of a meeting, (iii) the presentations that are scheduled for the meeting, (iv) the profiles of the speakers, and (v) the state of a meeting. To acquire this knowledge, the Context Broker explores different sources of information that is published on the Semantic Web and provided by the sensor agents (e.g., the Bluetooth Sensing Agent in Figure 8).

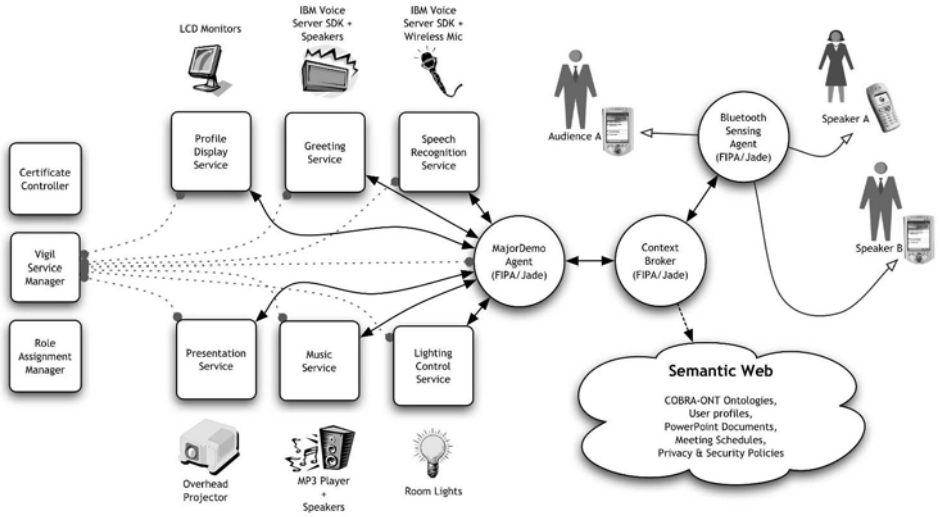

FIGURE 8. In EasyMeeting the Context Broker shares its contextual knowledge with the MajorDemo agent. Using this knowledge, MajorDemo selects and then invokes appropriate Vigil services to provide relevant services and information to the speakers and audiences.

The following is a typical EasyMeeting use case: Room 338 is a smart meeting room. On January 8th, 2004, a presentation is scheduled to take place from 1:00-2:30 PM in this room. Moments before the event starts, the room's Context Broker acquires the meeting's schedule from the Semantic Web and concludes the meeting is about to take place in the Room 338. As the meeting participants begin to arrive, the room's Bluetooth Sensing Agent detects the presences of different Bluetooth enabled devices (e.g., cellphones, PDA's). Because each device has a unique device profile that is described by standard Semantic Web ontologies, the sensing agent can share this information with the Context Broker.

Based on the user profile ontologies that are stored in the Context Broker's knowledge base (e.g., who owns what devices), without knowing any evidence to the contrary, the Context Broker concludes the owners of the detected devices are also located in the Room 338. Among the arrived participants, there are Harry (the speaker) and President Hrabowski (the distinguished audience). The Context Broker shares the location information of these participants with the subscribed MajorDemo agent.

Knowing that President Hrabowski has a distinguished audience role, the MajorDemo agent invokes the Greeting Service to greet him. At 1:00 PM, the Context Broker informs the MajorDemo agent that all listed *key* participants have arrived and that the presentation can be started. Knowing all the lights in

the meeting are currently switched on and the background music is also playing, the agent invokes the Dim Light Method on the the Light Control Service and the Stop Music Method on the Music Service.

As Harry walks to the front of the meeting room, he speaks to the system using a wireless microphone, "load Harry's presentation". The voice command is received by the Voice Recognition Service and a corresponding CCML command is generated. The MajorDemo agent sends this text string command to the Presentation Service along with the URL at which Harry's presentation can be downloaded (this information is provided by the Context Broker). As the Presentation Service loads Harry's PowerPoint slides, the MajorDemo agent invokes the Profile Display Service to show Harry's home page. Few seconds later, all LCD displays sitting on the conference table start showing Harry's biosketch and his profile. Using the same wireless microphone, Harry speaks to the system to control his presentation.

5.2. CoBrA Demo Toolkit

This toolkit a set of software applications for demonstrating the key features of CoBrA. It is aimed to provide a proof-of-concept demonstration and stimulate future system design and development. This toolkit has three key components: (i) a stand-alone Context Broker implementation in JADE, (ii) a customizable JADE agent called *ScriptPlay* for facilitating demo scripts, and (iii) an Eclipse Plug-in called CoBrA Eclipse Viewer (CEV) for monitoring the knowledge base changes in the Context Broker.

Using this toolkit, we can develop customized demonstrations to show how the knowledge base of a Context Broker changes when new contextual information is acquired or when the logic inference for context reasoning is triggered. Through a graphical user interface, users can (i) inspect the ontology schemas and data instances that form the Context Broker's belief about the present context, (ii) view privacy policies that the individual users have defined to protected their private information, and (iii) monitor the communication messages that are sent between the Context Broker and other agents.

CEV is a tool for browsing the context model in the knowledge base of the Context Broker and to monitor its changes while the Context Broker acquires new information from other agents or infers new context knowledge. Figure 9 shows a screenshot of the CEV plug-in that displays partial knowledge of the Context Broker.

One of our demonstration scenario is to show the Context Broker's ability to detect knowledge inconsistency when maintaining the location information of a person. In this demonstration, the ScriptPlay agent is configured to simulate a group of location tracking agents that individually send sensed people location information to the Context Broker. To simulate a real world scenario, some of the reported location information are intentionally made to be inconsistent with each other. For example, one report may express that a person is located in the Room ITE 325, which is part of the ITE building on the UMBC campus, and the other report may express the same person is located in some place in the state

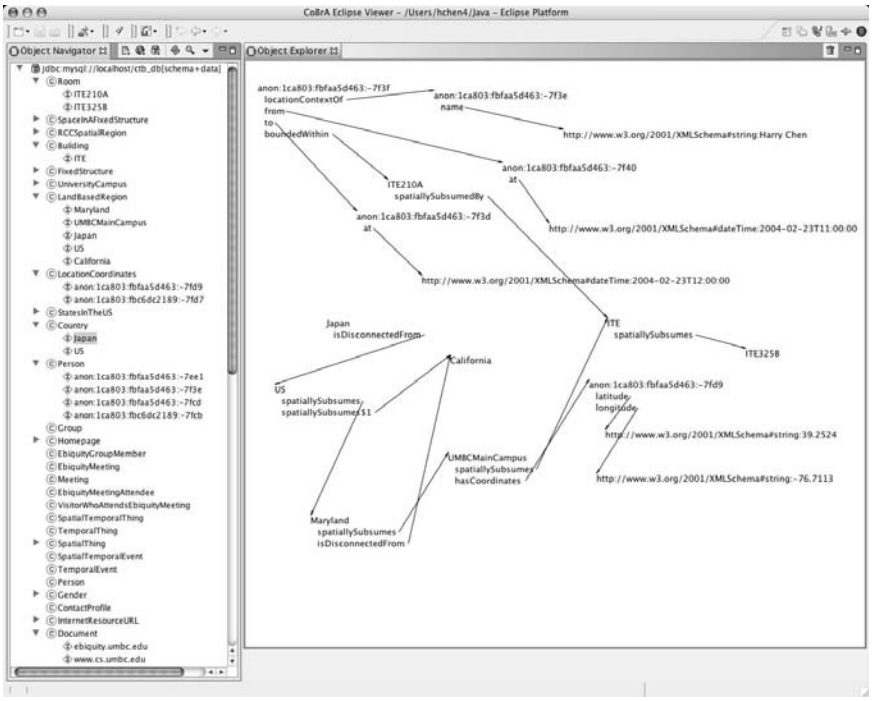

FIGURE 9. A screenshot of the CoBrA Eclipse Viewer (CEV). On the left, CEV lists the ontology classes and instances that are part the Context Broker's knowledge base. On the right, CEV users can explore the properties of individual class instances.

of California during the same time interval. Because the UMBC campus in in the state of Maryland, which is a geographical region that is disconnected from the state of California, the previous two reports of the same person's location context are inferred to be inconsistent.

In this demonstration, the Context Broker continuously waits to receive incoming reports about people's location context and notifies the senders when the reported information is inconsistent with its stored knowledge. If the incoming report is consistent with the stored knowledge, the Context Broker replies with a confirmation message and adds the new information to its knowledge base. Figure 10 shows a UML sequence diagram of a complete run of this demonstration[4]. At present, our implementation only handles the *detection* of inconsistent knowledge and does not handle the *resolution* of inconsistent knowledge. We are investigating different strategies for resolving inconsistent knowledge after it has been detected.

[4]A QuickTime video of this demonstration is available at `http://cobra.umbc.edu`

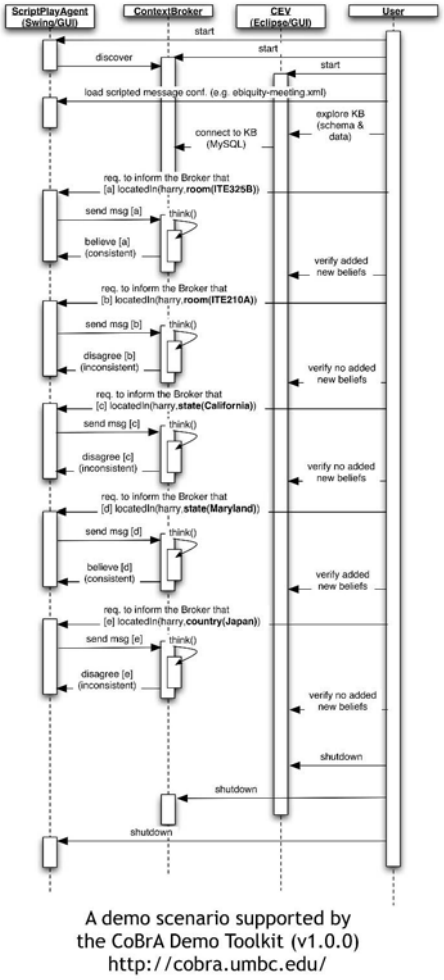

FIGURE 10. Using the CoBrA Demo Toolkit, users can monitor the underlying behavior and the knowledge base of the Context Broker and inspect the communication messages between the Context Broker and other agents. This demo shows the Broker's ability to detect inconsistent location information about a person when there are inaccurate sensing reports. Upon receiving information that is inconsistent with its existing belief, the Broker notifies the sender of the information and refuses to add this information to its knowledge base.

```
H1: locatedIn(Per,Rm), owner(Per,Dev) => locatedIn(Dev,Rm).

H2: locatedIn(Per,Rm), meeting(Mt,Rm),
    speakerOf(Per,Mt), not(notLocatedIn(Per,Rm))
       => intends(Per,give_prst(Mt)).

F1: locatedIn(t68i,rm338).
F2: owner(harry,t68i).
F3: meeting(m1203,rm338).
F4: speakerOf(harry,m1203).
```

FIGURE 11. Rules for assumption-based reasoning in the Theorist framework.

6. Future Work

Our near term objective is to improve the logic inference mechanism in the Context Knowledge Base. We are investigating the use of the *Theorist* framework [34], a Prolog meta-interpreter for processing assumption-based reasoning. Different from the conventional deductive reasoning systems, in this framework, the premises of the logic inference consists both facts (axioms given as true) and assumptions (instances of the possible hypotheses that can be assumed if they are consistent with the facts). Supporting both default reasoning and abductive reasoning is a key feature of the *Theorist* framework.

One way to use *Theorist* is for context reasoning, exploiting both default and abductive reasoning. In this approach, all contextual information acquired by the Context Broker are viewed as its observation about the environment. When an observation is received, the Context Broker first uses abduction to determine the possible causes and then uses default reasoning to predict what else will follow from the causes [28].

Let's consider the example in Figure 11. Hypotheses `H1` states that a personal device is located in a room if the owner of the device is also in that room. Hypotheses `H2` states that if a person is in a room where a meeting is scheduled to take place, the same person is the speaker of the meeting, and no evidence showing the person is not in that room, then the person intends to give a presentation at the meeting. Fact `F1` states that Cellphone T68i is located in the room RM338. Fact `F2`, `F3`, and `F4` state that Harry is the owner of the Cellphone T68i, Meeting m1203 is scheduled to take place in the room RM338, and Harry is the speaker of the Meeting m1203, respectively. We expect `F1` to be knowledge acquired from the sensors, and `F2`, `F3`, and `F4` to be knowledge acquired from the Semantic Web.

Our first objective is to infer the cause for the observation that the Cellphone T68i is located in the room RM338 (i.e., `F1`). We use abduction. Based on the given knowledge, {`locatedIn(harry,rm338), owner(harry,t68i)`} is a plausible explanation for `locatedIn(t68i,rm338)`. Knowing Harry is in room RM338, our second objective is to predict his intention in that room. We use default reasoning. Using `H2`, we can infer Harry intends to give a presentation in the Meeting m1203.

7. Conclusions

The Semantic Web languages and ontologies defined using these languages are of great importance to future pervasive context-aware systems. Using the OWL language we can define ontologies for modeling context and for supporting context reasoning. Shared ontologies can also help agents, services, and devices to share context knowledge and to interoperate in an open and dynamic environment. As the Semantic Web tools and ontologies emerge, they will bring new research opportunities in developing pervasive context-aware systems.

We have described the SOUPA ontology, an emerging standard ontology for supporting ubiquitous and pervasive computing applications. Based on our experience in prototyping CoBrA, the SOUPA ontologies have shown great promises in supporting context modeling, context reasoning, and knowledge sharing. The results of EasyMeeting and the CoBrA Demo Toolkit have successfully demonstrated aspects of the CoBrA system. In the EasyMeeting system, the Context Broker helped the smart meeting room services to provide relevant services and information to the meeting participants based on their context. In the demonstration supported by the CoBrA Demo Toolkit, we have shown the ability of the Context Broker to detect inconsistent information about people's location by reasoning with a set of geographical spatial ontologies. In the future, we will expand the demonstration to show the Context Broker's ability to resolve knowledge inconsistency and to protect the privacy of users.

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 recommendation 10 february 2004 edition, February 2004.

[2] Dan Brickley and R.V. Guha. Rdf vocabulary description language 1.0: Rdf schema. In *W3C Recommendation*. RDF Core Working Group, 2004.

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

[4] Roy Campbell, Jalal Al-Muhtadi, Prasad Naldurg, Geetanjali Sampemane1, and M. Dennis Mickunas. Towards security and privacy for pervasive computing. In *Proceedings of International Symposium on Software Security*, Tokyo, Japan, 2002.

[5] Jeremy Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: Implementing the semantic web recommendations. Technical Report HPL-2003-146, Hewlett Packard Laboratories, 2003.

[6] 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.

[7] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 2003.

[8] Harry Chen, Tim Finin, and Anupam Joshi. A context broker for building smart meeting rooms. In *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium*. AAAI, AAAI Press, March 2004.

[9] Harry Chen, Filip Perich, Dipanjan Chakraborty, Tim Finin, and Anupam Joshi. Intelligent agents meet semantic web in a smart meeting room. In *Proceedings of the Thrid International Joint Conference on Autonomous Agents & Multi-Agent Systems*, July 2004.

[10] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard ontology for ubiquitous and pervasive applications. Technical report, University of Maryland, Baltimore County, 2004. Submitted to The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004).

[11] 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.

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

[13] Simon Cox, Paul Daisey, Ron Lake, Clemens Portele, and Arliss Whiteside. Geography markup language (gml 3.0). In *OpenGIS Documents*. OpenGIS Consortium, 2003.

[14] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. *Lecture Notes in Computer Science*, 1995:18–??, 2001.

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

[16] Edd Dumbill. Finding friends with xml and rdf. In *IBM developerWorks, XML Watch*. xmlhack.com, June 2002.

[17] 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.

[18] Fabien L. Gandon and Norman M. Sadeh. Semantic web technologies to reconcile privacy and context awareness. *Web Semantics Journal*, 1(3), 2004.

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

[20] Jerry R. Hobbs. A daml ontology of time. `http://www.cs.rochester.edu/~ferguson/daml/daml-time-20020830.txt`, 2002.

[21] Lalana Kagal, Tim Finin, and Anupam Joshi. A Policy Based Approach to Security for the Semantic Web. In *2nd International Semantic Web Conference (ISWC2003)*, September 2003.

[22] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 2003.

[23] Lalana Kagal, Vlad Korolev, Harry Chen, Anupam Joshi, and Timothy Finin. Centaurus : A framework for intelligent services in a mobile environment. In *Proceedings*

of the International Workshop on Smart Appliances and Wearable Computing (IWSAWC), 2001.

[24] Lalana Kagal, Massimo Paolucci, Naveen Srinivasan, Grit Denker, Tim Finin, and Katia Sycara. Authorization and privacy for semantic web services. *AAAI 2004 Spring Symposium on Semantic Web Services*, March 2004.

[25] Lalana Kagal, James Parker, Harry Chen, Anupam Joshi, and Tim Finin. *Handbook of Mobile Computing*, chapter Security, Trust and Privacy in Mobile Computing Environments, pages ??–?? CRC Press, 2004.

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

[27] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, February 1990.

[28] Alan K. MacKworth, Randy G. Goebel, and David I. Poole. *Computational Intelligence: A Logical Approach*, chapter 9, pages 319–342. Oxford University Press, 1998.

[29] Deborah L. McGuinness and Frank van Harmelen. Owl web ontology language overview. `http://www.w3.org/TR/owl-features/`, 2003.

[30] Natalya Fridman Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory, 2001.

[31] Feng Pan and Jerry R. Hobbs. Time in owl-s. In *Proceedings of AAAI-04 Spring Symposium on Semantic Web Services*, Stanford University, California, 2004.

[32] Filip Perich. *MoGATU BDI Ontology*, 2004.

[33] 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.

[34] David Poole. Compiling a default reasoning system into prolog. *New Generation Computing*, 9(1):3–38, 1991.

[35] Shelley Powers. *Practical RDF*. O'Reilly & Associates, 2003.

[36] 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.

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

[38] Jeffrey Undercoffer, Filip Perich, Andrej Cedilnik, Lalana Kagal, Anupam Joshi, and Tim Finin. A secure infrastructure for service discovery and management in pervasive computing. *The Journal of Special Issues on Mobility of Systems, Users, Data and Computing*, 2003.

[39] 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.

[40] Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, June 1995.

Harry Chen, Tim Finin and Anupam Joshi
Dept. of CSEE
University of Maryland, Baltimore County
USA
e-mail: hchen4@cs.umbc.edu
finin@cs.umbc.edu
joshi@cs.umbc.edu