A Pervasive Computing Ontology for User Privacy Protection in the Context Broker Architecture

Harry Chen, Tim Finin, and Anupam Joshi
Department of Computer Science & Electrical Engineering
University of Maryland, Baltimore County

July 12, 2004

TR-CS-04-08

A Pervasive Computing Ontology for User Privacy Protection in the Context Broker Architecture

Harry Chen, Tim Finin, and Anupam Joshi

Department of Computer Science & Electrical Engineering
University of Maryland, Baltimore County
{hchen4,finin,joshi}@csee.umbc.edu

Abstract. Privacy protection is a key requirement for the future pervasive computing systems. This paper describes the design and implementation of a privacy protection framework that exploits the SOUPA policy ontology and its associated policy reasoning algorithm. The SOUPA policy ontology expressed in the Web Ontology Language OWL allows users to define policy rules to permit or forbid actions that attempt to access the users' private information. Central to the policy reasoning algorithm is the use of a Description Logic inference engine that reasons over the OWL-DL constructs of the policy ontology. We also show the feasibility of this framework through a prototype of the Context Broker Architecture (CoBrA).

1 Introduction

Privacy will be a great concern in the future pervasive computing systems. A vast amount of user information will be acquired and shared by different devices, services, and agents to provide users with relevant information and tailored services. Researchers [12, 3, 1, 11, 9] believe that in order to develop successful pervasive computing systems, we must also develop adequate infrastructures to protect the privacy of the users.

In this paper, we describe the design of a privacy protection framework and its implementation in the Context Broker Architecture (CoBrA). CoBrA [4] is a broker-centric agent architecture for supporting pervasive context-aware systems in smart spaces, e.g., smart meeting rooms, intelligent homes, and smart vehicles. Central to this architecture is an intelligent broker agent called *context broker* that maintains a share model of context for all computing entities and provides privacy protections for the users in the associated space. CoBrA uses the Web Ontology Language OWL to defined ontologies for supporting knowledge sharing, data fusion, and context reasoning.

CoBrA adopts a policy-based approach to protect user privacy. Policies are defined using the SOUPA policy ontology [6]. Using this ontology, users can define customized policy rules to permit or forbid different computing entities to access their private information. To compute the permissions defined by a user policy, the context broker is implemented with a policy reasoning algorithm. This algorithm exploits the use of a Description Logic (DL) inference engine and the DL constructs of the OWL language to decide whether an action for accessing some user private information is permitted or forbidden.

We believe the SOUPA policy language defined using the OWL-DL constructs can be used to develop intelligent agents that can provide user privacy protection in a pervasive context-aware environment. To show the feasibility of this privacy protection framework, we have implemented a CoBrA prototype that can support privacy protection use cases in an intelligent meeting room environment.

The rest of this paper is organized as follows. In Section 2, we review other policy languages that are designed for privacy protection. In Section 3, we describe the SOUPA policy ontology and its associated policy reasoning algorithm. In Section 4, we describe the privacy protection implementation in CoBrA and the supported use case scenarios. Conclusions and future works are given in Section 5.

2 Related Works

Policy is an emerging technique for controlling and adjusting the low-level system behaviors by specifying high-level rules [13]. The use of policy is common in computing systems that feature security or privacy protection [14]. In typical policy-based systems, policy rules are defined using declarative policy languages that are distinct from the actual system programming languages. A key advantage of using declarative languages to express policies is that the defined policies are more suitable for humans to view and edit. In addition, by separating the logics (i.e., policy rules) from the controls (i.e., programming implementations) of the system implementations, policy-based security and privacy protection systems are typically more flexible and adaptable than other non-policy-based systems [13].

Different research works in the policy domain often share a common vision on the usage of policy, i.e., using high-level rules to control low-level system behaviors. However, they usually adopt different representation languages to define policies. The choice of the policy representation language can affect the expressiveness and the flexibility of the defined policy language. For example, the Ponder policy language [8] has a representation that is similar to a typical procedure language, and therefore it is less expressive than other policy languages that have representations in meta-languages (e.g., XML and RDF) or Semantic Web languages (e.g., RDFS, DAML+OIL, and OWL). The P3P language has representations in both XML and RDF [7]. Allowing for more expressive constructs, the KAoS policy language is defined using DAML+OIL [2]. The Rei policy language [10] and the privacy policy language in the e-Wallet system [9] both adopt the OWL language as the representation language for policies.

The policy languages that are represented using the Semantic Web languages are usually defined in terms of ontologies. Different ontology organizations require different computing approaches to reason and analyze the defined policies. The design of the KAoS policy ontology suggests the use of a description logic inference engine to analyze policy rules. The Rei policy ontology, on the other hand, requires the use of an F-Logic based meta-interpreter (i.e., Rei/F-OWL) to compute the defined policy restrictions and constraints. The policy analysis mechanism in the e-Wallet system exploits the XSLT technology to translate policy rules from RDF to JESS rules and use a JESS rule engine to compute policy restrictions.

The SOUPA policy language described in this paper is similar to those ontology-based policy languages. It also exploits a Semantic Web language as the representation of policies. The SOUPA policy ontology is similar to Rei in modeling a policy as a set of rules that define restrictions on actions (e.g., an action with certain properties is permitted or forbidden), but the SOUPA policy ontology has limited support for meta-policy reasoning (e.g., conflict resolution) and speech acts (e.g., delegation and revocation). While the SOUPA and KAoS policy languages is represented using different Semantic Web languages, but they have similar policy reasoning mechanisms, both of which exploits the DL constructs of the corresponding ontology language.

3 The SOUPA Policy Ontology

3.1 About SOUPA

SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) [6] is a set of ontologies for supporting knowledge representation and knowledge sharing in the pervasive computing systems. The SOUPA project begins in November 2003 and is part of an ongoing effort of the Semantic Web in UbiComp Special Interest Group, 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 policy ontology described in this paper is part of the SOUPA ontology (version 2004-06). In addition to the policy ontology, SOUPA also includes ontologies for representing intelligent agents with associated beliefs, desires, and intentions, time, space, events, user profiles, and actions (see also `http://pervasive.semanticweb.org`).

3.2 The Design of the Policy Ontology

The SOUPA policy ontology is designed based on the following concept: *policies are rules that regulate the permissions for computing entities to perform actions*. In a pervasive computing environment, policies are defined by the human users to permit or forbid computing entities to perform different types of actions. The notion of an action can represent an invocation to some type of computing procedures to acquire user information or to access services in the computing environment. For example, actions can be calls to some low-level system API's, remote procedures, or web service interfaces, and actions also can be queries to some persistent data repository or communication messages to agents in the system.

We define an action with the following properties:

- **Actor**: an agent[1] that performs the action.
- **Recipient**: an agent that receives the effect after the action is performed.
- **Target**: a physical or an abstract object that the action applies to.

[1] Hereafter, the term "agent" will be used to represent an individual or a group of human agents, software agents, or any other types computing entities.

– **Location**: a place at where the action is performed.
– **Time**: a time at which the action is performed.
– **Instrument**: a physical or an abstract thing that the actor uses to perform the action.

When regulating the performance of actions, users can define policy rules to either *permit* or *forbid* actions to be performed. In a pervasive computing system, the enforcement of a policy can be the duty of a pre-established central authority or the obligation of all self-governing agents.

Policies are documents. All defined policies possess similar attributes that describe typical written documents (e.g., meta-data about the documents). The set of typical policy document attributes include the author who created the policy, the creation time of the policy, the version information of the policy, and the expected policy enforcer of the policy.

In a typical policy document, it is often infeasible to define explicit policy rules for every individual actions in a domain application. For example, in a location-aware system, a vast number of different agents may be interested to know the whereabouts of a user, it would infeasible for the user's policy to explicitly enumerate location information sharing rules for every one of the agents in the system. A solution to this problem is to define meta-policy reasoning behavior that can help the agents to reason about action permissions even when policy rules are not defined. The design of this policy ontology defines the following meta-policy reasoning behavior:

– **Conservative**: by default, assume all actions are forbidden. If no explicit rules are defined to regulate an action, then assume the action is forbidden.
– **Liberal**: by default, assume all actions are permitted. If no explicit rules are defined to regulate an action, then assume the action is permitted.

3.3 The Representation of the Policy Ontology

The OWL representation of the policy ontology is defined in a single ontology document under the XML namespace `http://pervasive.semanticweb.org/ont/2004/06/policy#`. This ontology imports the SOUPA time and action ontologies for representations of time and actions, respectively. For better readability, hereafter, we use the prefix `pol:`, `tme:`, and `act:` as the XML namespace shorthands for the SOUPA policy, time, and action ontology, respectively.

The OWL class `pol:Policy` represents a set of all policy documents. An individual of this class represents a policy document that regulates the permissions for agents to perform different actions. For a given `pol:Policy` class individual, the properties `pol:creator`, `pol:policyOf`, and `pol:enforcer` describe the agent who creates the policy, the agent whom this policy applies to, and the agent who enforces the defined policy rules, respectively. For describing the time when a policy document is created, the ontology defines the `pol:createdOn` property, and the range of which is the `tme:InstantThing` class. The individuals of the `tme:InstantThing` class are temporal descriptions of the time instants [6].

To describe policy rules, the ontology defines the property `pol:permits` for expressing permissions and the property `pol:forbids` for expressing prohibitions.

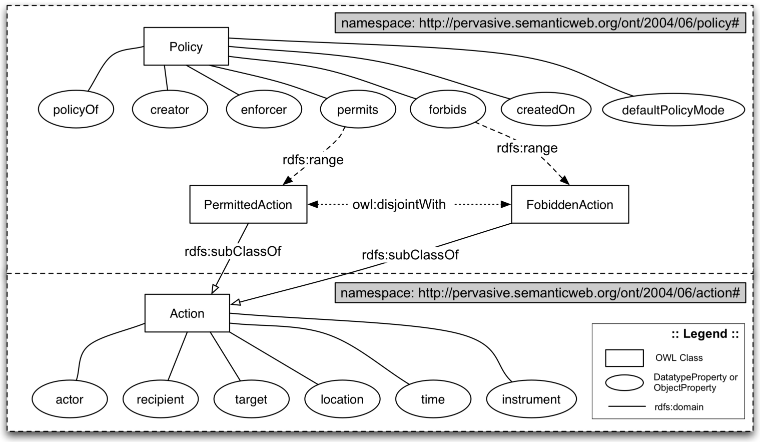

Fig. 1. The SOUPA policy ontology defines the vocabularies for describing rules that permit or forbid agents to perform different types of actions. The policy ontology imports the SOUPA action and time ontology for representations of action and time.

The domains of both properties are the `pol:Policy` class. The range of the `pol:permits` property is the `pol:PermittedAction` class, and the range of the `pol:forbids` property is the `pol:ForbiddenAction` class. Both action classes are subclasses of the `act:Action` class, but the set of individuals of each classes are disjointed.

The `pol:defaultPolicyMode` property is a meta-policy construct for specifying the policy reasoning behavior of a policy-enforcing agent. This property has domain `pol:Policy` and range `pol:Mode`. The `pol:Mode` class is an enumerated class, which consists of two pre-defined class individuals – `pol:Conservative` and `pol:Liberal`.

The OWL class `act:Action` represents a set of all actions. Within the SOUPA action ontology, six basic action properties are defined. They are `act:actor`, `act:recipient`, `act:target`, `act:location`, `act:time`, and `act:instrument`. The intended meanings of these properties are described in the previous section. Note that in the action ontology document, the respective range of these properties are unspecified, and they are intended to be specified in the subclasses that extend the `act:Action` class. We shall see some examples in Section 4.

3.4 An Algorithm for Policy Reasoning

In this section, we describe an algorithm for reasoning with the SOUPA policy ontology. The algorithm describes a procedure that can be implemented by a self-governing agent to compute its right to perform a specific action or implemented by a centralized authority to answer queries about the rights of other community agents to perform different actions.

Definitions. For an agent who is capable of reasoning with the SOUPA policy ontology, it is assumed to have the following components: (i) $\mathcal{KB}$: a knowledge base that stores the agent's world knowledge in RDF triples (SOUPA ontologies, contextual information, user policies, etc.), (ii) $\mathcal{R}_{DL}$: a Description Logic inference engine that support the OWL-DL reasoning, (iii) $f_{action}(I)$: a function that maps an agent's intention I (i.e., an action that an agent intends to perform) into a set of RDF triples, and (iv) $f_{policy}(A)$: for a given action description A, this function outputs a list of user policy documents that each of which contains some rule that regulates the execution of the action.

We assume the outputs produced by the $f_{action}(I)$ and $f_{policy}(A)$ functions are expressed in the SOUPA ontologies and ontologies that extend from SOUPA. We also assume the users are responsible to provide the agent with their policy documents.

Exception Handling. The objective of the algorithm is to help a policy-enforcing agent to effectively decide, for a given set of associated user policies and an intended action, whether the execution of the action is permitted or forbidden by the policies. Exception conditions may occur while the agent is performing policy reasoning. First, an exception can occur when there is no available user policy in the $\mathcal{KB}$. Second, an exception can occur when the rules in a user policy neither permit nor forbid the execution of an action. Lastly, an exception can occur when one or more user polices contain conflicting rules that permits and forbids the execution of an action simultaneously.

In this algorithm, the first exception is handled by assuming the existence of a global policy, which is adopted by the policy-enforcing agent by default. To handle the second exception, the algorithm makes use of the meta-policy construct (i.e., `pol:default-PolicyMode`) to decide the permission of an action when it is not regulated by any policy rules. For the last exception, the algorithm delegates the exception handling task to the upper-level agent implementation by "flagging" the detection of policy inconsistency.

The Algorithm. Algorithm 1 defines a procedure called COMPUTE-PERMISSION. It takes $\mathcal{KB}$, $\mathcal{R}_{DL}$ and I as the inputs and returns a value that represents the computed permission to perform an action A. Note that A is the output of $f_{action}(I)$. Let $\mathrm{P\_TYPE}$ be an enumeration type, which has a set of values PERMIT, FORBID, and $\mathrm{UNDECIDED}$. The values of $\mathrm{P\_TYPE}$ represent all possible computed permission values that can be returned by COMPUTE-PERMISSION.

In Algorithm 1, the procedure DO-CLASSIFICATION (line 9) is called to execute the associated DL classification functions of $\mathcal{R}_{DL}$, and the procedure LIST-INDIVIDUAL-CLASS-TYPES (line 10) is called to return the names of all ontology classes of which the action A is an instance after the classification is made. The utility procedure USE-POLICY-DEFAULT-BEHAVIOR (line 14) helps the agent to compute permissions when actions are not explicitly regulated by any policy rules. After computing the action permission for each user policy, the procedure FORGET-ASSERTATIONS (line 27) is called to execute the associated DL functions to remove all assertions about the current policy from the inference engine, so that the inferences of each policy document are independent.

Algorithm 1 Computes the permission to perform an intended action I

```
COMPUTE-PERMISSION(KB, R_DL, I)
perm ← UNDECIDED
A ← f_action(I)
P[] ← f_policy(A)
if length[P] = 0 then
    P[1] ← *global_policy*
end if
for i = 1 to length[P] do
    perm_tmp ← UNDECIDED
    DO-CLASSIFICATION(R_DL, KB, P[i], A)
    T[] ← LIST-INDIVIDUAL-CLASS-TYPES(R_DL, A)
    if CONTAINS(T, PermittedAction) = TRUE and
    CONTAINS(T, ForbiddenAction) = TRUE then
        error "inconsistent policy"
    else if CONTAINS(T, PermittedAction) = FALSE and
    CONTAINS(T, ForbiddenAction) = FALSE then
        perm_tmp ← USE-POLICY-DEFAULT-BEHAVIOR(P[i])
    else if CONTAINS(T, PermittedAction) = TRUE then
        perm_tmp ← PERMIT
    else if CONTAINS(T, ForbiddenAction) = TRUE then
        perm_tmp ← FORBID
    end if
    if perm_tmp = PERMIT and perm = FORBID then
        error "inconsistent policy"
    else if perm_tmp = FORBID and perm = PERMIT then
        error "inconsistent policy"
    else
        perm ← perm_tmp
    end if
    FORGET-ASSERTATIONS(R_DL,P[i])
end for
return perm
```

```
USE-POLICY-DEFAULT-BEHAVIOR(policy)
behavior ← POLICY-DEFAULT-BEHAVIOR(policy)
if behavior = CONSERVATIVE then
    return FORBID
else if behavior = LIBERAL then
    return PERMIT
end if
```

4 Privacy Protection in CoBrA

At UMBC, a CoBrA prototype has been used to facilitate context-awareness in a smart meeting room system called EasyMeeting [5]. The goal of this system is to create a smart meeting room that can provide relevant information and tailored services to the users of an everyday meeting room. The use of contextual information is an important element in this meeting room system [5]. In EasyMeeting, a context broker is responsible to acquire and reason about contextual information that is associated with meeting events, and then to share its knowledge with other agents to provide context-aware services. The type of contextual information used in this system includes the location of people and devices, the schedule of meeting events, the profiles of the event speakers and audiences, and the user intentions and beliefs that are inferred by the context broker.

Although the feedbacks from the EasyMeeting user experience studies were positive, many users showed great concerns for privacy [5]. In this rest of this section, we examine key privacy issues in EasyMeeting, and show how the SOUPA policy ontology and its associated policy reasoning are incorporated into the new CoBrA prototype to provide privacy protection.

4.1 Privacy Issues in the EasyMeeting System

In EasyMeeting, let's assume that users are willing to share some of their personal information with the computing environment in order to receive tailored meeting services. One problem is that the users may be unaware that the services, which have received their private information, may also share the same information with other services. For example, the speaker of a meeting is willing to share his/her location information and profile with the context broker, so that the context broker can notify the appropriate presentation agent to set up his/her presentation. However, the same presentation agent may in turn share some or all of the speaker's information with other services without informing the speaker.

Another problem is that depending on the context of the users, users' trust in allowing services to access their private information can vary. In other words, users may not always trust the same service to access their private information. For example, meeting attendees may be willing to share their contact information and location information with the smart space services while they are attending the meeting. However, after the meeting is over, the same meeting attendees may have less trust in allowing the services to use their contact and location information.

In addition, there is a problem relating to the granularity of the information being shared. Agents often need to access users' private information in order to provide relevant services. In a pervasive computing environment, while users want to hide much of their private information from the untrusted agent, often they cannot completely prohibit all information to be share if they desire to receive context-aware services. In other words, if the EasyMeeting system is to adopt a privacy protection mechanism, it must allows users to share information with the computing services at different granularity levels. This way a balance between protecting user privacy and enabling context-aware services can be achieved.

```
# For a complete and more elaborate policy example, see also
#       http://cobra.umbc.edu/ont/2004/05/harrychen-policy

<http://umbc.edu/~hchen4/hchen.pol> a pol:Policy;
  pol:policyOf  [ a per:Person; per:name "Harry Chen"^^xsd:string ]
  pol:defaultPolicyMode pol:Conservative;
  # Rule 1: all individuals of CLS2 are permitted actions#
  pol:permits ha:CLS2;
  # Rule 2: all individuals of CLS3 are forbidden actions#
  pol:forbids ha:CLS3.

ha:CLS2 a :Class;
   rdfs:comment "Share my location information with the ebiquity group members iff
                 the location information describes me being in ITE210A, ITE325B or
                 on the UMBC campus.";
   :intersectionOf  (
      ebact:ShareLocationInfo
      [ :allValuesFrom ebm:EbiquityMember; :onProperty act:recipient ]
      [ :onProperty act:target; :someValuesFrom ha:MyRestrictedLocationContext ] ) .

ha:CLS3 a :Class;
   rdfs:comment "Share my location information with untrusted service agent";
   :intersectionOf  (
      ebact:ShareLocationInfo
      [ :allValuesFrom ha:UntrustedServiceAgent; :onProperty act:recipient ] ) .

ha:MyRestrictedLocationContext a :Class;
  :intersectionOf  (
     loc:LocationContext
     [ :onProperty loc:boundedWithin; :someValuesFrom ha:foo-a1 ] ) .

ha:foo-a1 a :Class;
  :oneOf  (ebgeo:ITE210A  ebgeo:ITE325B ebgeo:UMBCMainCampus) .

ha:UntrustedServiceAgent a :Class; rdfs:subClassOf agt:Agent;
    :oneOf  (
        <http://www.orbitz.com#locTrack> <http://www.foobar.com#whereRu>
        <http://www.foofoobar.com#abc>  ) .
```

Fig. 2. An example of the SOUPA policy. This policy permits some of Harry Chen's location information to be shared with the members of the eBiquity Group, and forbids any location information to be shared with the untrusted agents. The action class `CLS2` and `CLS3` describes the types of actions that the individual policy rule regulates.

4.2 Protect Privacy Using the Policy Ontology

To address the described privacy issues, we incorporated the use of the SOUPA policy ontology and its associated reasoning algorithm in the CoBrA prototype. We demonstrated the use of policies to protect two different types of user information – (i) personal profiles and (ii) location information. Personal profiles consist of information that describes users' the contact information, social networks, professional backgrounds, and personal devices. Location information is information that describes the whereabouts of the users.

In our implementation, all user information is described using the SOUPA ontologies and the COBRA-ONT ontologies [4]. Extending from the general concepts in SOUPA, COBRA-ONT consists of ontologies that are specially designed for supporting smart space services for the UMBC eBiquity Group's weekly meetings.

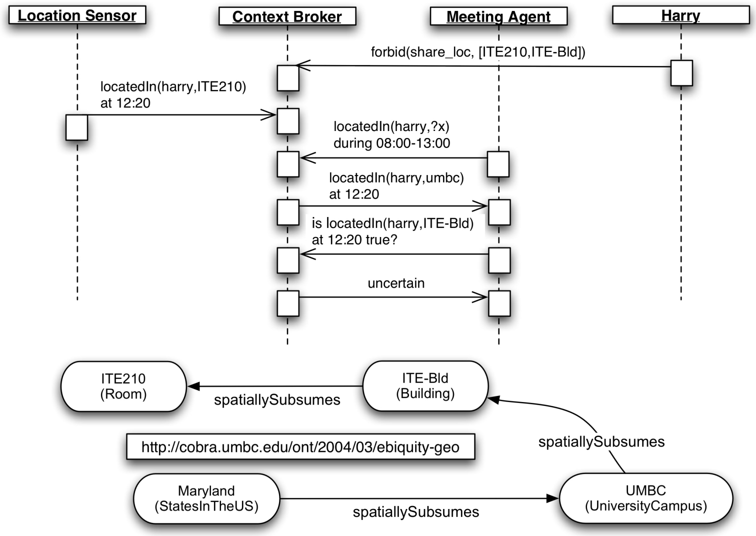

Fig. 3. The context broker exploits spatial reasoning to develop a less detailed version of the forbidden location information and shares that information with the meeting agent to maximize the benefit of knowledge sharing. The spatial reasoning procedure analyzes the spatial hierarchy that is defined by the `spc:spatiallySubsumes` property.

Harry Chen's Privacy Policy. The same set of ontologies for expressing user information can also be used in expressing user policy. Figure 2 shows a partial policy of the person Harry Chen (note: all RDF/OWL ontology meta-data is omitted here for simplicity). Two rules are defined in this policy. Based on the range definitions in the SOUPA policy ontology, Rule 1 expresses that the action class `CLS2` is type of `pol:PermittedAction`, and Rule 2 expresses that the action class `CLS3` is type of `pol:ForbiddenAction`.

Reason with Harry's Policy. The policy reasoning procedure of the context broker implements Algorithm 1. In our implementation, when the context broker receives a service agent's request to access a user's private information (e.g., Harry's location information), it will attempt to retrieves the user's privacy policy (i.e, Harry's policy) from its knowledge base. This policy is sent to the context broker priorly by the user's personal agent. If no associated policy is found, a default policy is used. Based the incoming request (i.e., the intention of the requesting agent), the context broker generates a RDF/OWL description of the action (i.e., an individual of the `act:Action` class). Using a DL reasoner[2], the action individual is then classified within the SOUPA

[2] In our implementation, we use the Jena inference API that is backed by a Racer reasoner through the DIG inference. See also `http://jena.sourceforge.net/how-to/dig-reasoner.html`

and the COBRA-ONT ontologies. If the DL reasoner infers that the action individual is type of `pol:PermittedAction`, then the context broker will answer the request with the appropriate user information. On the other hand, if the action instance is type of `pol:ForbiddenAction`, then the context broker will deny the request.

Proximity in Policy Reasoning. Sometimes simply denying an incoming request to access user information can be too restrictive. For example, assuming that the context broker has knowledge that Harry is currently located in the Room ITE-210A on the UMBC campus, however, it refuses to tell a meeting coordinating agent because Harry's policy prohibits the sharing of his location information at the "room" level. In a worse case scenario, not knowing Harry is at UMBC, the meeting coordinating agent cancels a meeting that Harry has previously scheduled. To overcome this problem, we have implemented a proximity reasoning technique that allows the context broker to infer and share a less detailed version of the forbidden user location information. This can help to maximize the benefit of knowledge sharing. Figure 3 shows a use case of this proximity reasoning technique.

5 Conclusions & Future Works

Privacy protection will be a key requirement for the future pervasive computing systems. The use of policy can be an effective mechanism to allows users to take control of their private information in an open and dynamic environment. We believe that the Semantic Web languages such as OWL and RDF/RDFS are suitable languages for defining new policy languages because of their expressive power and support for knowledge representation and reasoning and knowledge sharing and integration.

In the new CoBrA prototype[3], we demonstrated the use of policy to protect user information that is typically exploited by the smart meeting room applications. We also showed that the SOUPA policy ontology, which is defined using the OWL-DL constructs, and its associated algorithms can be used to develop intelligent agents that can provide user privacy protection in a pervasive context-aware environment.

In the future, we plan to address the following issues related to policy management and reasoning: (i) in order to help users to manage their policy, we must design an intuitive interface for editing and validating the SOUPA policy documents. Possible solutions may be extending the existing ontology editors (e.g., Protege or OilEd) to include customized policy editing and validating function, or creating a policy editor plug-in application for the Eclipse IDE. (ii) We plan to design and implement new algorithms that will help agents to reason about the dependences among different user information. Using these algorithms, the context broker can further protect user privacy by prohibiting information to be shared if it allows others to infer information that is currently forbidden by the policy. (iii) We plan to develop a Java API for composing, storing, and reasoning policies expressed in the SOUPA policy ontology. By extending the Jena API, this new software library can help developers to quickly create policy-aware agents that adopt the SOUPA policy protection framework.

[3] Source codes and other information can be found at `http://cobra.umbc.edu/`.

References

[1] Victoria Bellotti and Abigail Sellen. Design for privacy in ubiquitous computing environments. In *Proceedings of the Third European Conference on Computer Supported Cooperative Work (ECSCW'93)*, pages 77–92. Kluwer, 1993.

[2] Jeffrey M. Bradshaw, Andrzej Uszok, Renia Jeffers, Niranjan Suri, Patrick J. Hayes, Mark H. Burstein, A. Acquisti, Brett Benyo, Maggie R. Breedy, Marco M. Carvalho, David J. Diller, Matt Johnson, Shriniwas Kulkarni, James Lott, Maarten Sierhuis, and Ron Van Hoof. Representation and reasoning about daml-based policy and domain services in kaos. *Proceedings of The Second International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS2003)*, 2003.

[3] 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.

[4] 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*, 18(3):197–207, 2004.

[5] 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.

[6] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard ontology for ubiquitous and pervasive applications. In *Proceedings of the First International Conference on Mobile and Ubiquitous Systems: Networking and Services*, 2004.

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

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

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

[10] 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.

[11] 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.

[12] Marc Langheinrich. Privacy by design – principles of privacy-aware ubiquitous systems. In G.D. Abowd, B. Brumitt, and S. Shafer, editors, *Proceedings of Ubicomp 2001*, volume 2201 of *Lecture Notes in Computer Science*, pages 273–291. Springer, 2001.

[13] Morris Sloman and Emil Lupu. Security and management policy specification. *IEEE Network, Special Issue on Policy-Based Networking*, 2002.

[14] Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, 2003.