Policy-based Access Control for Task Computing Using Rei

Ryusuke Masuoka, Mohinder Chopra, Yannis Labrou, Zhexuan Song, Wei-lun Chen
Fujitsu Laboratories of America
8400 Baltimore Avenue, Suite 302
College Park, MD 20740, USA
+1 (301) 486-0398
{ryusuke.masuoka, mohinder.chopra, yannis.labrou, zhexuan.song, sam.chen}@us.fujitsu.com

Lalana Kagal
Massachusetts Institute of Technology (MIT)
Computer Science and Artificial Intelligence Laboratory (CSAIL)
32 Vassar Street, Cambridge, MA 02139, USA
+ 1 (617) 253-2613
lkagal@csail.mit.edu

Timothy Finin
University of Maryland, Baltimore County
1000 Hilltop Circle
Baltimore, MD 21250, USA
+1 (410) 455-3522
finin@umbc.edu

ABSTRACT

In this paper, we describe a policy-based access control implementation for Task Computing using the Rei policy engine.

Task Computing lets ordinary end-users accomplish complex tasks on the fly from an open, dynamic, and distributed "universe of network-accessible *resources*" in ubiquitous computing environments as well as those on the Internet.

The Rei policy specification language is an expressive and extensible language based on Semantic Web technologies. The Rei policy engine reasons over Rei policies in OWL and domain knowledge to answer queries about the current permissions and obligations of an entity.

To provide unobtrusive and flexible access control for Task Computing, a framework was created in which several Rei policy engines were endowed with Web Services APIs to dynamically process facts from clients, the private policies of service providers, shared policies, and common shared ontologies. The framework is implemented and deployed for Fujitsu Laboratories of America (FLA), College Park office and evaluated.

Categories and Subject Descriptors

J.7 [**Computer Applications**]: Computers in Other Systems

General Terms

Management, Design, Experimentation, Security, Human Factors, Languages.

Keywords

Task Computing, Rei, Policy, Semantic Web, OWL, OWL-S

1. Introduction

As the World Wide Web evolves as a computing and network infrastructure, policy management becomes crucial to provide access control not only for information on the Internet, but resources in general, including networked devices and Web Services, in such diverse environments as ubiquitous computing and grid computing.

This paper focuses on access control for end-users to resources in ubiquitous computing environments. These resources are described abstractly in OWL as services, and are mainly realized as UPnP devices and simple Web Services. This focus of resources in ubiquitous computing poses a different set of requirements and problems than for information on the Internet. It is not that one is more difficult than the other. For example, a framework can leverage from the physical reality in ubiquitous computing environments. But the very dynamic nature of ubiquitous computing environments definitely offers new kinds of challenges.

Our contribution is to add a flexible policy-based access-control to ubiquitous computing and demonstrate its utility and effectiveness in a ubiquitous computing application. Task Computing (TC, [1][2][3][[4]]) is a user-oriented framework that lets end-users accomplish complex tasks on the fly from open, dynamic, and distributed "universe of network-accessible *resources*" in environments rich with applications, devices, and services. Task Computing provides many ways for users to interact with these ubiquitous environments and applies Semantic Web technologies, such as OWL (Web Ontology Language, http://www.w3.org/2001/sw/WebOnt/) and OWL-S (http://www.daml.org/servcies) as its core enablers. In each environment, functions (devices/OS/applications) are virtualized as services. Through discovery mechanisms such as UPnP, TC clients find those services and obtain their OWL-S files as their semantic service descriptions. With those OWL-S files, TC clients let the end-users to manipulate (compose, execute, publish, etc.) the services on the spot.

When started, the TC client dynamically finds the local services on the computer on which it runs and pervasive services in the sub-network the computer is on. UPnP is used for the service discovery on the sub-network. When a local or subnet service is discovered, the TC client downloads the appropriate OWL-S files that represents its semantic service description. Using the OWL-S descriptions, TC client such as STEER allow a user to compose and execute the services. The user can also create new semantic services dynamically by instantiating or composing other services. For example, Task Computing enables a user to display a presentation file from his mobile PDA or computer on the stationary room projector without connecting a VGA cable, even if this is the first time the user has been in the room. In another example, a user can print a presentation file from his laptop on the printer provided in the room without configuring his computer, or show a photo just taken with his digital camera on the photo frame in the same room immediately and print it on a photo printer without moving memory cards around, or, display the current weather at an address in his PIM (Personal Information Manager) on the projector with just a few operations of point and click. Task Computing enables end-users to accomplish all of the above and more through a simple graphical user interface to the Task Computing environment. You can even use your own voice to make those same things happen through a voice-based Task Computing client, VoiceSTEER.

Rei is a policy specification language for describing different kinds of policies in a wide range of application domains. The main goal of Rei is to address the issue of governing *autonomous* entities in *constantly evolving* distributed environments. Rei provides specifications for describing declarative machine-understandable policies enabling both policy enforcement and a more normative approach where autonomous entities can decide whether or not to fulfill the applicable policy.

Rei is represented in an extension of OWL-Lite ([6][8][9][10]) and can be used to describe policies over domain knowledge in different ontology languages such as RDF, DAML+OIL, and OWL. Though its classes and properties are represented in OWL, Rei also includes logic-like *variables* giving it the flexibility to specify constraints that are not directly possible in OWL e.g., the uncle relation, the same age as relation etc. Rei models deontic concepts of permission, prohibition, obligation, and dispensation and supports speech acts such as delegation, revocation, cancel, and request for dynamic policy modification.

As most entities in pervasive environments will have several overlapping policies of behavioral norms, constraints, and rules acting on them, they will be over-constrained. This means that they cannot always satisfy all of the policies, but deviating too much or too often has its consequences - loss of reputation, penalty clauses, imposition of sanctions, etc. Rei provides two mechanisms for handling these situations namely consequences and meta policies. Rei allows consequences to be modelled as 'sanctions' so that autonomous entities or providers can reason over them to decide whether or not to deviate from a certain policy. Rei also allows meta policies to be used to resolve conflicts. Rei models two main types of meta policies: (i) for defaults and (ii) for conflict resolution to handle different requirements of policies. Depending on the type of conflict resolution required, the appropriate meta policy should be selected. Some policies may want a more high level meta policy and can use default behaviors or modality precedences. However, for tighter control, priorities are more suitable but are tougher to define and maintain.

In order to support policy development, Rei provides two forms of policy analysis: use-cases (also known as test-case analysis) and what-if analysis (also known as regression testing). The policy engine includes analysis tools accessible via a Java interface that can be executed by policy engineers to check the consistency and validity of the policies and ontologies.

From the initial implementation of Task Computing Environment, it was immediately apparent that it requires some kinds of access control for the services because it makes so easy for the end-users to use the devices and services dynamically found on the same sub-network. In home network environment, it would not be so much a problem as long as the network is firewalled from the outside networks. But when Task Computing should be applied to, for example, office or hospital environments where there are many devices that should be protected from abuses by unauthorized accesses.

To provide unobtrusive and flexible access control for Task Computing, a framework is created with Rei policy engines endowed with Web Services API to process facts from the client, service's private policy, shared policies, and ontologies dynamically. The framework is implemented and deployed for Fujitsu Laboratories of America (FLA), College Park office and evaluated.

In this paper, the motivation and design goals of the work are given in Section 2. The implementation and test deployment of Task Computing access control with the Rei policy engine is described in Section 3. Then Section 4 describes how the above design goals are met. Related work is discussed in Section 5 and Section 6 concludes this paper.

2. Motivation and Design Goals

As mentioned above, the initial implementation of Task Computing immediately revealed the need for access control of services. A simple access control mechanism, which will not be described any further here, was implemented in the early stage. This mechanism leveraged the physical embodiment as devices of many services in Task Computing and this mechanism is often enough for a simple deployment of services based on devices, but it had its limitations. It was inappropriate for large deployments of dynamic services and clients, or for services *without* their physical embodiments. Simple identity or role based access control mechanisms were unable to meet the requirements of these dynamic environments. A sophisticated policy-based solution for Task Computing was necessary to cover such cases. At the core of the solution, a way to express rule based policies and an engine to process the policies were required. The Rei policy specification language and Rei policy engine came as a perfect match.

Rei is an expressive policy language based on Semantic Web technologies. As Task Computing had already embraced OWL and OWL-S as its core enabler, it made it easier to integrate many aspects of Task Computing into the policy language. Specifically Task Computing needs seamless inferences over policies, facts, and ontologies. The Rei policy engine can combine dynamically policies including delegations, OWL ontologies, and facts described using ontologies and infer the access rights for users and programs.

The dynamic nature of ubiquitous computing environments also requires the policies to be defined not in terms of ID’s and roles, but rules based on properties of entities such as users, devices, and services. In the ubiquitous environment with often unforeseeable entities, the access control should shift to rule-based approaches using descriptions of entities involved.

In order to give enough flexibility, it necessitates the use of mechanisms for updating the policies on the fly. Especially delegation mechanisms, which Rei also supports, are imperative. Users do let others use devices and services on their behalf or temporarily in everyday life. If the system does not allow such flexibility, the users would be forced to drop the mechanism totally or find a way to evade it.

We also deem it important that the system allows developers, system administrators, and even end-users to specify the policies in a natural and intuitive way. It would make the system very easy to use if, for example, the system lets the user specify policy very close to everyday languages and processes them in the way the ordinary people would expect it to be processed. While the policy language itself needs not to have everyday language aspects, a policy language with high expressivity enables such a system by allowing mapping the user's policy specifications correctly into the policy language.

Such considerations made the Rei policy specification language and policy engine a natural choice for us.

On the other hand, we wanted to get leverage from the ubiquitous computing environments as application areas of Task Computing. It can be difficult to hand out credentials signed by appropriate CA's to the users. As it turns out in the next section, the process is smoothly incorporated into the office check-in process and the credential is copied on the physical memory device for the user with the full Task Computing client on it. The credential is sent by the user through the Task Computing client to the service to be authenticated and consumed by the Rei engine.

When we design the access control for Task Computing Environment using Rei, the following items are set as its design goals.

1. Minimally obtrusive for users
2. Enable both centralized/distributed solutions
3. Allow multiple certificate authorities
4. Secure dynamic delegation

For the first point, it is always a trade-off between security and ease of use. (You can create a perfectly secure system … just let no one use it.). But with appropriate technologies and smart deployment of the system, we can shift the balance, more security with less obtrusiveness. If the access control is difficult or cumbersome to use, it would kill the Task Computing experience. It is also imperative to finish the policy calculation in a reasonable amount of time. The access control is secondary function to the main function. It is like putting the cart in front of the horse if it takes longer time than the main function.

Secondly, we wanted to have both centralized/distributed solutions possible because the access control deployments can be different from one site to another. For some site, an IT department might want to manage the policy centrally, thus requiring a centralized solution. In some other cases, the end-user might want to set some policy for a single device. It is preferable, for example, the end-user can set the policy for the device at its initial configuration. Such a distributed solution is often enough for a small office. There is another aspect of centralized/distributed solutions as to where the policy engine should run. In case of resource-limited devices, there might be no choice, but to choose the centralized solution in which the device accesses the policy engine running on a different more powerful machine.

The third point is important when you consider the applications for relatively open spaces such as shopping malls. By allowing multiple certificate authorities in the framework, it can maximize the chances that the user can use the service. Of course, the user and the service need to agree on at least one common certificate authority that they both trust, in order for authentication to happen.

The last point is crucial in order to make the access control flexible. Sometimes one wants to override the default access control to let someone else to use the service. It is necessary that the person has the enough authority to do it and that it should be done securely. But if the system does not allow such flexibility, the user would eventually find the system useless or tries to find ways to circumvent the access control.

3. Implementation and Test Deployment

We have ported the Rei policy engine to run in the Windows environment because many of the Task Computing services are provided by Windows-based systems. A Web Services API was created for the Rei engine to facilitate its use in a highly distributed environment. We incorporated the access control based on the Rei policy engine into the "*Pervasive Print*" TC service, which lets users print files remotely (without any printer setup) to create the "*Secure Print*" service. The Credential Creator software was produced to easily create a digitally signed credential in the OWL format. We also created the Delegation Manager software to let the users insert and/or remove delegation statements (in the Rei format) into/from the shared policy site securely over HTTPS.

The resulting system was deployed in the Fujitsu Laboratories of America (FLA), College Park office. The Credential Creator was installed on the desktop machine in the reception area, the *"Secure Print"* service was installed on a computer with a printer in the conference room along with the Rei policy engine. (Here we had the "distributed solution" in the sense that the policy engine is distributed to each service.)

We will explain the usage scenario first and then give the details how it is realized.

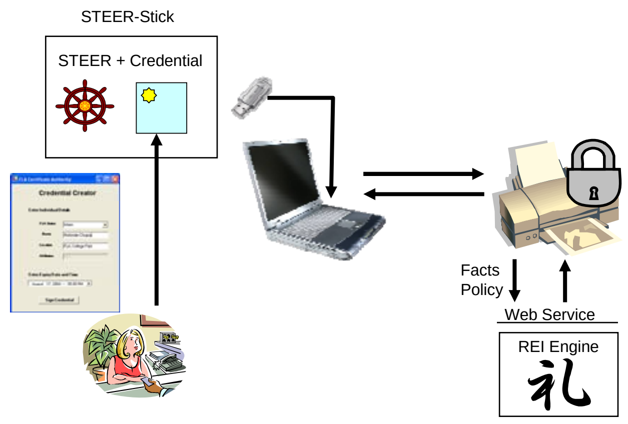

Figure 1. Deployment of Task Computing Access Control

The scenario goes like this (See Figure 1). Mohinder, a UMBC (University of Maryland, Baltimore County) student, visits FLA, College Park. Valerie, the Office Administrator of FLA, College Park, greets Mohinder in the reception area.

1. Valerie creates a STEER-Stick with credential for Mohinder.

STEER-Stick is a USB memory device with all the software necessary to run STEER, a Task Computing client including Java runtime along with the credential. The credential includes his name, affiliation, status (*''Visitor''*) and metadata of credential (its creation date, expiration date/time, etc.), and the digital signature signed with the FLA private key. The Credential Creator software saves the credential in OWL format in the credential folder of the STEER-Stick. It also saves an HTML file for the human to check the contents of the credential.

✓ Mohinder runs the STEER using the STEER-Stick in his laptop. STEER finds the "Secure Print" service dynamically and show the service with a key icon.

The "*Secure Print*'' OWL-S file states that an FLA credential is required. (It can state that it requires one of multiple credentials.) When STEER finds that the service requires a credential, it shows a key icon for the service.

2. Mohinder tries to use the "*Secure Print*", but he fails because a "*Visitor*" is not allowed to use it.

Based on the "*Secure Print*" OWL-S file, STEER looks for an FLA credential in its "credential" folder. When it finds it, it sends the credential along with service invocation parameters in the Web Service call.

"*Secure Print*" checks the digital signature of credential to make sure it is valid. (So that facts in the credential are not modified.) Then it uses these facts to determine if the caller has the authority to use the service by the Rei policy engine, which is called through Web Service API. The Rei policy engine determines that Mohinder can not use the service as he is just a "Visitor" and returns the result through its Web Service API. The service , in turn, returns an error for the original Web service call with the reason.

3. Mohinder asks Ryusuke to delegate the right to print.

Ryusuke uses the Delegation Manager software to assert a statement to delegate the right to Mohinder by Ryusuke to the FLA shared policy site securely.

4. Mohinder tries again to use the "Secure Print" and this time he succeeds.

There is a statement at FLA Policy Site that Senior Employee has a right to delegate the right to visitors. With the newly added statement of the delegation, it enables Mohinder to print.

5. After that, Ryusuke revokes the delegation.

The delegation assertion created in the step 3 is removed from the FLA shared policy site by using the Delegation Manager. Mohinder can not use the service any longer.

Access control is determined based on the following elements:

✓ Facts provided by the client (authenticated by the digital signature)

✓ Printer's private policy

✓ FLA shared policy (and potentially other shared policies)

✓ Ontologies

The service can use multiple shared policies depending on its configuration. Each time, these elements listed above are mixed to determine the access control.

Figure 2 shows what happens behind the scene. The number given in the figure corresponds to the numbered item in the scenario. Shared policies and ontologies are cached and they are downloaded only when they are updated.

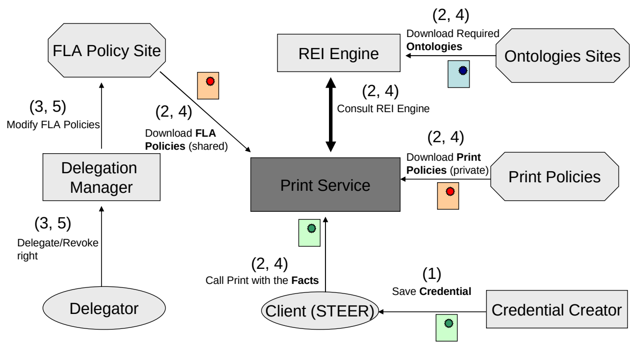

Figure 2. What is Happening behind the Scene

Figure 3 gives parts of fact, private policy for the Secure Print service, and the shared policy for FLA used in this scenario.

The scenario above centers around the value for "flaonto:Status" in the fact. All pieces of information in the fact are digitally signed and the digital signature assures its integrity. If any part of it is changed, the facts can not be authenticated.

Another thing to note is that it has the expiration time as a part of the credential's metadata. If the time has passed this expiration time, the "*Secure Print*" service will decline any request to print.

The Printer's private policy states that it can be used by a senior employee, but not by a visitor. Therefore Mohinder, who is a visitor, fails to print at first.

The FLA shared policy states that a Senior Employee has the right to delegate the right to use the "*Secure Print*" service (It is not shown in Figure 3). When Ryusuke insert his delegation statement (which is shown in Figure 3) using the Delegation Manager, this enables Mohinder to use the "*Secure Print*" because the service detects the update at the FLA Policy site and downloads the new FLA shared policy (and because of the statements that Ryusuke Masuoka is a Senior Researcher and that a Senior Researcher is a Senior Employee in the ontologies).

```
<!– Fact from Task Computing client -->
<rdf:RDF …>
  <rdfs:label lang=en>Mohinder Chorpa</rdfs:label>
  <flaonto:Name  …>Mohinder Chorpa</flaonto:Name>
  <flaonto:Expiry …>2004-08-23T23:05:28Z</flaonto:Expiry>
  <flaonto:Status …>&flaonto;FLACPVisitor</flaonto:Status>
  <flaonto:Affiliation …>UMBC</flaonto:Affiliation>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      …
    </SignedInfo>
    <SignatureValue>ZrbEVA7JWWGNbpqc…Jo6dDw=</SignatureValue>
  </Signature>
</rdf:RDF>

<!– Printer Private Policy -->
…
<deontic:Permission rdf:about="&flapolicy;right_to_be_printed_on“
  policy:desc="All senior employees have the right to print">
  <deontic:actor rdf:resource="&flapolicy;var1"/>
  <deontic:action rdf:resource="&flapolicy;printing_in_conference"/>
  <deontic:constraint rdf:resource="&flapolicy;preOrSenior"/>
</deontic:Permission>
…

<!– Delegation Inserted (and Removed) in Shared Policy-->
<action:Delegation
  rdf:ID=“Delegation2004-08-23T19:32:19ZRyusukeMasuoka">
  <action:sender rdf:resource="&inst;RyusukeMasuoka"/>
  <action:receiver rdf:resource="&inst;MohinderChorpa"/>
  <action:content>
    <deontic:Permission>
      <deontic:action rdf:resource="&inst;ASeniorEmployeePrintingAction"/>
    </deontic:Permission>
  </action:content>
</action:Delegation>
```

Figure 3. Fact, Private Policy, and Shared Policy

The example given above is kept relatively simple for the sake of easy understanding. The system as it is now can fully utilize the expressivity which the Rei engine allows. For example, a scenario, such as one in which a senior employee gives to a class of users (ex. all visitors from UMBC on Jan 31st) the right to use a class of resources (ex. all devices in the conference room), is possible.

4. Evaluation and Discussions

In this section, we discuss how we met the initial design goals set forth in Section 1.

1. Minimally obtrusive for users

We tried to keep the additionally requirements for all the users involved as little as possible.

We have created software tools such as Credential Creator and Delegation Manager so that end-user needs not to write complex OWL/Rei statements, but just to give essential information.

The credential creation process is integrated into an ordinary office check-in process in which the Office Administrator types in the visitor’s name and affiliation, selects the appropriate status (selections created dynamically from an ontology) and the expiration time in the Credential Creator GUI ,and hit the save button. The digitally signed credential in OWL is automatically created and saved in the appropriate folder of the STEER-Stick USB memory device,. The STEER-Stick includes a full Task Computing client, STEER, on it and the user can run STEER from the STEER-Stick without any installation.

STEER hides the details of using the secured services and shows only essential information. Secured services are shown with key icons so that the user knows that it requires appropriate authority to execute it. When the execution fails because of the security clearance, it will notify the user the reason. All the details are handled behind the scene such as determining from the OWL-S file if the service is secured and what kinds of credential is necessary and sending appropriate credential to the secured service.

On the service side, we found that the Rei policy engine needed to be accelerated so as not to hamper the user’s experience. Originally it took seven to eight seconds to finish the access control calculation based on the fact, policies, and ontologies. In general, caching answers does not help as we can not expect fact, policies, and ontologies to remain fixed (especially facts). We made various changes to the Rei policy engine to enable it to produce answers to queries in less than one second.

2. Enable both centralized/distributed solutions

From the aspect of policy management, we can have the spectrum between centralized and distributed solutions. One can put the policies that should be kept private in the private policy while policies that can or should be shared can be put in one of the shared policies at the shared policy sites. Which shared policies for the service to use is up to the service to decide.

From the aspect of policy engine, the Rei policy engine with Web Services API allows very flexible deployment as long as the Rei policy engine is accessible from the service by HTTP/HTTPS. But the privacy of private policy is compromised to some degree when the Rei policy engine is running on a different machine because the private policy needs to be sent to the Rei policy engine for the access control calculation.

3. Allow multiple certificate authorities

We allow the OWL-S file for the service to include the multiple certificate authorities that the service accepts. On the other hand, STEER looks into its credential folder for credentials from compatible certificate authorities for the service and send the credential along with the Web Services calls if found.

For example, Mohinder may carry two credentials, one from FLA and one from UMBC in the credential folder. The OWL-S file of “Secure Print” may state that it requires a credential from FLA or 8400 Baltimore Avenue Building (where FLA, College Park office is located in). STEER selects the credential from FLA in the credential folder to use “Secure Print” service.

4. Secure dynamic delegation

With the Delegation Manager software, it is possible for end-users easily to insert (and later remove) the Rei delegation assertions into the shared policy hosted at a Web server securely over HTTPS. This gives flexibility often necessary in everyday usage of the system.

In addition to the initial design goals, we would like to discuss here about our decision not to make the Rei engine Web Services discoverable dynamically as a semantic service as it is usually the case for Task Computing Web Services. While it is easy to make the Rei Web Service discoverable through, for example, UPnP and the service automatically starts using the Rei Web Service, it can be a security hole simply doing that. The dynamically found Rei engine needs to be authenticated and there is a bootstrapping issue. It is also likely that the human service provider has a very specific idea of which policy engine to use along with each service.

5. Related work

Extensible Access Control Markup Language (XACML) [16] is a language in XML for expressing access policies. This work is similar to ours; in that it allows control over actions and supports resolution of conflicts. However, as it is based in XML, it does not benefit from the interoperability and extensibility provided by Semantic Web languages. It also does not model speech acts or handle conflict resolution across policies.

Lately there has been a significant body of standardization efforts for XML-based security, such as WS-security, -trust, and -policy at W3C, or SAML of the OASIS Security Services Technical Committee, and the Security Specifications of the Liberty Alliance Project. The standards support low-level security or policy markups that concern formats of credentials or supported character sets for encoding. They do not address semantic user- or application-specific trust tokens and their relations. These standards have been developed to support controlled B2B applications where both client and service can be mutually authenticated and recognized. These standards are not extensible to more dynamic environments in which simple authentication is not enough, but authentication on user-defined attributes needs to be considered. For this, a semantic approach like we take in this paper, is a possible solution.

KAoS is a policy language based in OWL [17][18]. This language is similar to Rei in that it can be used to develop positive and negative authorization and obligation policies over actions. KAoS policies are OWL descriptions of actions that are permitted (or not) or obligated (or not). This limits the expressive power, so that there are policies that Rei can describe that KAoS cannot. However, KAoS allows the classification of policy statements enabling conflicts to be discovered from the rules themselves. The Rei engine can only discover conflicts with respect to a particular situation and cannot pre-determine them. However, Rei includes run-time conflict resolution by supporting meta-policies.

The paper [19] presents an XML-based specification language, which incorporates content and context based requirements for documents in XML-based Web Services. It uses a role-based access control model which simplifies authorization administration by assigning permissions to users through roles. Although it relates roles to permissions, there is no way to dynamically change these roles or permissions. Using the delegation module of REI we can change the policies dynamically to adapt to the changes in roles or permissions.

6. Conclusion

It is our belief that security and access control should be natural, flexible and minimally obtrusive for the end-users as they try to accomplish everyday tasks. If not, the users will eventually find ways to evade the mechanisms rendering them useless, at best, and possibly counter-productive. It is also important to give enough flexibility in the deployment aspect of security and access control because their requirements and rules differ from one site/office to another.

To that regard, we have been successful in adapting our flexible access control framework to blend in an ordinary office environment.

Future work includes:

✓ Discovery security

By making it so that only accessible services are found for Task Computing client, it will make the whole system more secure and easy to use.

✓ Service authentication

By using the OWL-S file of the service, the service notifies its (shareable part of) policy to the client. It enables the client to better determine if the service is executable in advance.

✓ Explanation and negotiation

The user would get frustrated if the system simply rejects his/her use of certain resources without giving a reason. The system needs to give out understandable explanation for the rejection if asked. It should also be very useful if the system can provide the information on what it requires in order to gain permission.

7. REFERENCES

[1] Masuoka, R., Parsia, B., and Labrou, Y., "Task Computing – The Semantic Web meets Pervasive Computing -." In D. Fensel et al. (Eds.), "The Semantic Web - ISWC 2003," the Second International Semantic Web Conference (ISWC 2003), Sanibel Island, FL, USA October 2003 Proceedings, LNCS 2870, 2003, pp. 866-881.

[2] Masuoka, R., Labrou, Y., Parsia, B., and Sirin, E., "Ontology-Enabled Pervasive Computing Applications," IEEE Intelligent Systems, vol. 18, no. 5, Sep./Oct. 2003, pp. 68-72.

[3] Song, Z., Labrou, Y., and Masuoka, R., "Dynamic Service Discovery and Management in Task Computing," MobiQuitous 2004, August 22-26, 2004, Boston, USA, pp. 310 - 318.

[4] Task Computing Home Page, http://taskcomputing.org

[5] Lalana Kagal, Tim Finin, and Anupam Joshi, "Trust Based Security for Pervasive Computing Enviroments", IEEE Communications, December 2001.

[6] Lalana Kagal, "Rei : A Policy Language for the Me-Centric Project", HP Labs Technical Report, 2002.

[7] Jefferey Undercoffer, Filip Perich, Andrej Cedilnik, Lalana Kagal, Anupam Joshi, 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.

[8] Lalana Kagal, Tim Finin, and Anupam Joshi, "A Policy Language for Pervasive Systems", Fourth IEEE International Workshop on Policies for Distributed Systems and Networks, 2003.

[9] Lalana Kagal, Tim Finin and Anupam Joshi, "A Policy Based Approach to Security for the Semantic Web", Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003.

[10] Lalana Kagal, "A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments", Dissertation, September, 2004.

[11] Grit Denker, Lalana Kagal, Tim Finin, Massimo Paolucci, and Katia Sycara, "Security for DAML Web Services: Annotation and Matchmaking", Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003,

[12] Lalana Kagal and Tim Finin, "Modeling Conversation Policies using Permissions and Obligations", AAMAS 2004 Workshop on Agent Communication (AC2004), July, 2004,

[13] Pranam Kolari, Lalana Kagal, Anupam Joshi, and Tim Finin, "Enhacing P3P Framework with Policies and Trust", UMBC Technical Report and under review, 2004.

[14] Anand Patwardhan, Vlad Korolev, Lalana Kagal, and Anupam Joshi, "Enforcing policies in Pervasive Environments", International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004.

[15] Lalana Kagal, Massimo Paolucci, Naveen Srinivasan, Grit Denker, Tim Finin, and Katia Sycara , "Authorization and Privacy in Semantic Web Services", IEEE Intelligent Systems (Special Issue on Semantic Web Services), 2004.

[16] S. Godik and T. Moses, "OASIS eXtensible Access Control Markup Language (XACML)", OASIS Committee Secification cs-xacml-specification-1.0, November 2002.

[17] A. Uszok, J. Bradshaw, P. Hayes, R. Jeffers, M. Johnson, S. Kulkarni, M. Breedy, J. Lott, and L. Bunch, "DAML reality check: A case study of KAoS domain and policy services", International Semantic Web Conference (ISWC 03), Sanibel Island, Florida, 2003.

[18] A. Uszok, J. M. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton, and S. Aitken, "Policy and Contract Management for Semantic Web Services", AAAI Spring Symposium, First International Semantic Web Services Symposium, 2004.

[19] Arif Ghafoor, James B. D. Joshi, Rafae Bhatti, and Elisa Bertino, "XML-Based Specification for Web Services Document Security", IEEE Computer Society Press , 2004.