DRAFT
Towards a Well-Behaved Web

Lalana Kagal
University of Maryland
Baltimore County
CSEE Department
1000 Hilltop Circle
Baltimore, MD 21250
lkagal1@cs.umbc.edu

Tim Finin
University of Maryland
Baltimore County
CSEE Department
1000 Hilltop Circle
Baltimore, MD 21250
finin@cs.umbc.edu

Anupam Joshi
University of Maryland
Baltimore County
CSEE Department
1000 Hilltop Circle
Baltimore, MD 21250
joshi@cs.umbc.edu

ABSTRACT

In order to realize the full potential of the World Wide Web as an open and dynamic network of information, it is important to govern how web entities (e.g., web services and agents) behave in terms of what resources they access (security), how their information is used by others (privacy), whether they are reliable (trust), and how they establish and fulfill social and business obligations and contracts (obligation management).

We propose that a *declarative policy-based approach* be used, where the norms or rules of ideal behavior of web entities are described in a machine-understandable specification language and web protocols are modified to include policy exchange, negotiation, compliance checking, and possibly enforcement. Web entities can define policies for several aspects of both their own behavior and the expected behavior of entities they will interact with, including security, privacy, collaboration, and commitments. These policies can be easily updated causing the behavior of the entities to be modified but without affecting the underlying protocols, mechanisms, or architecture. Along with providing the openness required, these policies also provide greater autonomy as they help interacting entities understand each others capabilities, requirements, limitations, and obligations and infer what their ideal behavior should be and act accordingly.

In this paper, we describe Rei, a policy specification language represented in an extension of OWL-Lite, which can be used to describe and regulate different kinds of behavior in a wide range of domains. Along with using a rule-based approach for greater expressivity, it also models several aspects of social policies including *consequences of violating policies* and *conditional permissions* that grant certain permissions on the condition that the authorized entities take on certain additional responsibilities. We illustrate the applicability of Rei for policy management of the web through two prototype applications namely (i) web privacy, and (ii) security, privacy, and confidentiality for semantic web services.

1. INTRODUCTION AND MOTIVATION

The Web is a dynamic network of information that has to accommodate a wide range of domain knowledge due to diverse organizational boundaries, adapt to heterogeneous and semi-autonomous entities, and manage variations caused by ambiguous boundaries and permutable services and resources. It enables entities to interact and share information and resources dynamically across organizations and domains. However, the probability of entities misbehaving is high leading to low reliability and trust. This reduces the usefulness of the web as entities are less willing to share information if they do not have control over who uses their information and how it is used, stored, and distributed. In order to realize the full potential of the World Wide Web as an open and dynamic network of information, it is important for entities to be able to understand the behavior of entities they interact with in terms of access control, fulfilling their obligations, accountability, etc. As the web is inherently decentralized and as entities are semi-autonomous, each entity should be able to describe how it behaves and the behavior it expects from entities it will interact with, and there should be appropriate decentralized mechanisms for matching descriptions of interacting entities and for ensuring (or verifying) that the behavioral specifications are (or were) followed.

Until recently, research in regulating the behavior of entities has been focused on distributed environments that were fairly static. In this model, though clients, services, and data were physically distributed, they were static, pre-determined and shared the same domain knowledge. In general, we believe that traditional mechanisms for governing behavior have the following characteristics that make them inappropriate for the web : (i) They are inflexible as they are tightly integrated with the security mechanisms. This implies that they can be only used in a restricted set of domains and their applicability to dynamic environments with fluctuating boundaries that involve a range of different domains is limited. (ii) They are not very expressive; in general, they cannot express all aspects of resources, users, and context. For example, RBAC usually allows permissions and responsibilities to be associated only with roles and does not take any other attributes into consideration. (iii) They are unable to adapt quickly to changes or provide schemes for dynamic modification. Few mechanisms provide any automated schemes for run-time modification such as delegation and revocation. (iv) They tend to decrease the autonomy of the entities as the actions of entities are actually restricted according to the security specifications without giving them the choice to govern their own behavior.

The characteristics of the Web that make behavior man-

.

Characteristic	Restriction	Solution
Resources and clients cannot be pre-determined	Scheme should not include hard-coded mechanisms that list access rights of individuals	Declarative policies described over different attributes of entities
The web is constantly evolving	Run-time modification should be possible	Speech acts are used for dynamic modification of policies
Entities are not always known or cannot always be authenticated	Scheme should not be based solely on identity of either the client or the service	Policies are based on attributes associated with entities and the environment
Very large number of resources, services, and clients	It should be possible to control sets of entities	Group policies are possible
No central control or repository	Decentralized control should be possible	Decentralized control possible where every entity follows its own policy
Presence of semi-autonomous entities	Scheme should be understandable and usable by humans and semi-autonomous entities	Policy specification language is represented in machine-understandable ontology language
The web spans several domains	Scheme should be usable for any domain-specific information	Policy engine can reason over different domain knowledge described in ontology languages
Entities are overconstrained	Scheme should provide mechanism for either resolving conflicts or for deciding which policy to deviate from	Policy language provides meta policies for automated conflict resolution and models sanctions allowing entities to understand the penalties of violating a policy

Table 1: Characteristics of the Web

agement difficult along with the reasons for our proposed solution are highlighted in Table 1. We propose that *declarative policies*, represented in a machine understandable format, be used to guide the behavior of web entities and web protocols be modified to include policy exchange, negotiation, compliance checking, and fulfillment. The policy framework should include support for dynamic policy modification in order to adapt to constantly changing requirements and meta policies to help resolve policy conflicts. The policy system should also include support for more autonomous web entities like web services and agents.

In this paper, we describe Rei, a policy specification language and framework, for rule-based regulation of entities in dynamic distributed environments [22, 19]. We propose that Rei be used for describing policies of web entities and the Rei policy engine be used to understand the behavior of entities and for checking compliance and fulfillment of policies. Rei provides a novel combination of six features that makes it suitable for the web: (i) it provides the extensibility required as policies can be described over different kinds of domain knowledge at different levels of abstractions, (ii) it can describe both positive and negative authorization and obligation policies that can be used to appropriately model different kinds of behavior, (iii) it includes a policy engine and analysis tools, (iv) it uses a rule-based approach and allows policies to be described in terms of attributes of users, actions, and other context, (v) it supports meta-policies for automated conflict resolution, and (vi) it supports dynamic policy modification via speech acts.

2. REI POLICY MANAGEMENT SYSTEM

The Rei policy language, represented in OWL-Lite, allows different kinds of policies (security, privacy, conversation, etc.) to be specified as restrictions over allowable and obligated actions in terms of attributes of the actor, action and the general context. 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.

As most entities on the web 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 allows these consequences to be modeled as 'sanctions' so that autonomous entities or providers can reason over them to decide whether or not to deviate from a certain policy. These 'sanctions' describe the penalties of violating the policy and are attached to prohibitions and obligations. A 'sanction' is capable of expressing common penalties of violating policies in human societies : retracting a permission, granting an additional obligation, or reducing a reputation measure. Rei also provides meta-policies for automated conflict resolution.

Rei can be used to express conditional permissions, which are common in human societies. These permissions allow an entity to perform a certain action or set of actions under the condition that it will take on certain additional responsibilities. These conditional permissions impose additional obligations on the entity after the permission is exercised. For example, *You can use the BravoAir service if you fill out the comments form.*

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

Rei has been used for policy management is several dynamic distributed applications including, (i) authorization in supply chain management systems [23, 24], (ii) access control in pervasive computing environments [21, 34], (iii) security,

privacy, and confidentiality for semantic web services [6, 25], (iv) collaborative multi-agent system [4], (v) web privacy [26], (vi) enforcing domain policy on handheld devices [32], (vii) security for Fujitsu's Task Computing project [27], and (viii) modeling conversation policies in multi-agent systems [20].

Though Rei has been mostly used for controlling what actors can/must do, it is also possible to model behaviors like "digital rights management", which typically focus not so much on actors can/must do but on objects and what can/must be done to/with them. For example, *the purchaser of this song is permitted to make copies that can only be played on a computer on which his private key is installed*. Rei can be used to represent and reason over DRM policies and even licenses like the Creative Commons Licenses [1].

2.1 Specifications

Rei specifications include policies, deontic objects, constraints, actions, speech acts, and policy analysis.

2.1.1 Policy

Rei policies are basically rules of behavior described over different attributes of the requester, service/action and the context. They are defined by a set of deontic objects where the relation that links the deontic concepts to the policies is 'grants'. The property *grants* can either be a deontic object or a granting.

Consider a student policy for the *CS Department*. It states that students have the permission to print on black and white printers. It uses a previously defined deontic, *Perm_StudentPrinting*, and adds a constraint that checks whether the printer is black and white using the *requirement* property.

```
<!-- Rei variables -->
<entity:Variable rdf:ID="#PersonVar"/>
<entity:Variable rdf:ID="#ActionVar"/>
<entity:Variable rdf:ID="#TargetVar"/>

<constraint:SimpleConstraint rdf:ID="IsCSStudent"
     constraint:subject="#PersonVar"
     constraint:predicate="&rdf;type"
     constraint:object="&univ;CSStudent"/>

<constraint:SimpleConstraint rdf:ID="GetTargetOfAction"
     constraint:subject="#ActionVar"
     constraint:predicate="&action;target"
     constraint:object="#TargetVar"/>

<constraint:SimpleConstraint rdf:ID="IsBWPrinter"
     constraint:subject="#TargetVar"
     constraint:predicate="&rdf;type"
     constraint:object="&univ;BlackWhitePrinter"/>

<constraint:And rdf:ID="And_Constraint">
     <constraint:first rdf:resource="#GetTargetOfAction"/>
     <constraint:second rdf:resource="#IsBWPrinter"/>
</constraint:And>

<policy:Granting rdf:ID="Granting_PermToBWPrinter">
     <policy:to rdf:resource="#PersonVar"/>
     <policy:deontic rdf:resource="#Perm_StudentPrinting"/>
     <policy:requirement rdf:resource="#And_Constraint"/>
</policy:Granting>

<policy:Policy rdf:ID="CS_Policy">
     <policy:context rdf:resource="#IsCSStudent"/>
     <policy:grants rdf:resource="#Granting_PermToBWPrinter"/>
</policy:Policy>
```

[1] http://creativecommons.org/

Also associated with a policy is its *context*, which defines the environment under which the policy is applicable as a set of additional constraints. In the above case, the context states that the actor must be a CS student.

2.1.2 Granting

A granting adds a set of constraints to an existing deontic object to form a new policy rule. This allows *reuse of deontic objects* in different policies with varied constraints and actors and helps *modularize* the policy making process. More technical policy makers can develop deontic objects over applicable domain action with generic constraints attached. Then domain administrators will create policies by integrating the pre-defined deontic concepts and adding more domain-specific constraints. For example, the following policy:Granting uses an existing deontic:Permission and defines additional constraints on the actor and actor. The following granting example gives all PhD students the permission to use laser printers where 'IsLaserPrinterAndPhD' is a boolean constraint.

```
<policy:Granting rdf:ID="Granting_StudentLaserPrinting">
     <policy:to rdf:resource="#PersonVar"/>
     <policy:deontic rdf:resource="#Perm_StudentPrinting"/>
     <policy:requirement rdf:resource="#IsLaserPrinterAndPhD"/>
</policy:Granting>
```

2.1.3 DeonticObject

This class is used to create rules that make up policies. It includes constructs for describing what action (or set of actions) the deontic is described over, who the potential actor (or set of actors) of the action is, and under what conditions is the deontic object applicable. Consider as an example, a deontic object that permits students to print on certain printers

```
<deontic:Permission rdf:ID="Perm_StudentPrinting">
     <deontic:actor rdf:resource="#PersonVar"/>
     <deontic:action rdf:resource="#ActionVar"/>
     <deontic:constraint rdf:resource="#IsStudentAndBWPrinter"/>
</deontic:Permission>
```

As the deontic:actor and deontic:action are both entity:Variables, the permission is applicable to all entities and actions that satisfy deontic:constraint. Two optional properties, *startingConstraint* and *endingConstraint* can be used together to define the window within which the deontic object is valid or must be completed in case of an obligation. DeonticObject has 4 subclasses : *Permission*, *Prohibition*, *Obligation* and *Dispensation*. Permission has an additional property called *provision*, which is an obligation. It is used to grant conditional permissions. The defined obligation only becomes active after the permission is exercised. For example, *You can use my car, if you return it with a full gas tank*. Prohibitions and Obligations have a *sanction* property associated with them that describes the penalties that will be imposed on the actor if the prohibition or obligation is violated. A sanction is usually an action or a set of actions that may be performed if the prohibition is violated. For example, *You are prohibited from cheating in an exam. If you do you will be suspended*.

Other deontic objects including power, claims, and liability can also be modeled in the Rei framework but are currently not included.

2.1.4 Constraint

A constraint is used to define sets of objects like the set of graduate students, the set of actions whose targets are laser printers, the set of users that belong to the same lab as 'JohnDoe', and the set of students whose advisors are assigned to the lab in which the target of the printer is located. There are two subclasses of constraints: *SimpleConstraint* and *BooleanConstraint*.

2.1.4.1 SimpleConstraint.

As any expression in RDF is a collection of triples, a SimpleConstraint is modeled as a triple consisting of subject, predicate and object.

For example, a set of objects of that have 'affiliation' property set to 'CSDept' is defined as

```
<constraint:SimpleConstraint rdf:ID="IsMemberOfCS">
     <constraint:subject rdf:resource="#PersonVar"/>
     <constraint:predicate rdf:resource="&univ;affiliation"/>
     <constraint:object rdf:resource="&univ;CSDept"/>
</constraint:SimpleConstraint>
```

2.1.4.2 BooleanConstraint.

Constraints (both Simple and Boolean) can be combined in pairs using operators of *And*, *Or*, and *Not* (which is assumed to be negation as failure). Each operator is a subclass of BooleanConstraint. For example, an 'And' operation over two constraints can be defined as

```
<constraint:And rdf:ID="IsLaserPrinterAndPhStudent">
     <constraint:first rdf:resource="#IsPhDStudent"/>
     <constraint:second rdf:resource="#IsLaserPrinter"/>
</constraint:And>
```

2.1.5 MetaPolicy

As we are developing Rei as a policy language for dynamic environments, we believe that meta policies are very important as the likelihood of conflicts between policies in these environments is high. Meta policies describe how policies are interpreted and how conflicts are resolved. Rei models two main types of meta policies to handle different requirements of policies (i) defaults and (ii) conflict resolution. Depending on the conflict resolution required, the appropriate meta policy should be selected. Some policies may require a higher 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.

2.1.5.1 Defaults.

These include behavior and meta-meta policies. Behavior is made up of three instances that describe what the default behavior or mode of the policy is, whether everything that is not explicitly permitted is prohibited, everything that is not explicitly prohibited is permitted, or everything has to be explicitly defined. As there can be more than one meta policy defined, In order to resolve a conflict between meta policies, a meta-meta policy can be set that decides the order in which meta policies are invoked.

2.1.5.2 Conflict Resolution.

Meta policies for conflict resolution include priorities and modality precedence. Priorities can be set either between policies and rules. For example, you can say that the Federal policy always overrides the State policy in case of conflict and you can also state that rule1 has greater priority that rule2 in the Federal policy.

It is also possible to set preferred modality e.g. negative modality is preferred for the CS department policy. So if a student has both a permission and a prohibition, the negative rule i.e. the prohibition will be given preference.

2.1.6 Speech Acts

Rei defines four speech acts namely *Delegation, Revocation, Request* and *Cancel* that are primarily used for dynamic modification of policies. Delegations are very important to dynamic environments because the entities may not be able to project who will use them or pre-establish all the desirable requirements of the entities who should use them. Delegations allow permissions to a resource to be propagated to a set of trusted entities, without explicitly changing the policy. Consider a web service that is only accessible by users of a certain role in a certain organization. These users also have the permission to delegate the ability to use this service to other users in the organizations for a certain time period. A delegation from one of these users to another employee of the organization permits the latter to use the web service without any explicit modification of the policy or code of the service as long as the constraints associated with the delegation are met. Thus a valid delegation leads to a new permission. Similarly, a revocation speech act nullifies an existing permission (whether policy based or delegation based) causing a prohibition. An entity can request another entity for a permission, which if accepted causes a delegation, or to perform an action on its behalf, which if accepted causes an obligation. An entity can also cancel any previously made request, which leads to a revocation and/or a dispensation.

As an example, consider a delegation speech act from George to Marty giving Marty the permission to use a service, *BravoAir ReservationAgent*.

```
<action:Delegation rdf:ID="Del_GeorgeToMarty">
     <action:sender rdf:resource="#George"/>
     <action:receiver rdf:resource="#Marty"/>
     <action:deontic>
          <deontic:Permission>
               <deontic:actor rdf:resource="#Marty"/>
               <deontic:action rdf:resource=
                         "&bravo-service;BravoAir_ReservationAgen
          </deontic:Permission>
      </action:deontic>
</action:Delegation>
```

As with all actions, an entity must have the permission to perform a speech act in order for it to be valid or allowed. This means that in order to perform a delegation that adds a permission, the entity must have the permission to delegate. For example, *Susan, a professor delegates to Alice, her student, the permission to write a review for a paper and also delegates the permission to further delegate the permission to write the review to any other student belonging to the same group.* This implies that Alice now has the permission to delegate the permission to review Susan's paper. The initial permission to delegate and the following speech acts add constraints to the sender of the speech act, the recipient, the action and to how the permission can be further delegated. Speech acts are tightly integrated into the framework as they affect policies that in turn affect speech acts. For example, *Tim revokes George's permission to delegate the permission to use BravoAir_ReservationAgent.*

2.2 Tools

Along with providing an expressive policy language, Rei also includes a policy engine and analysis tools.

2.2.1 Policy engine

We have developed a policy engine that reasons over policies in Rei, domain information and the context to answer queries about the current permissions and obligations of entities in the environment. It is able to answer several types of queries including, (i) Does X have the permission to perform Y on resource Z, (ii) What are the current obligations of X, (iii) What actions can X perform on resource Z, (iv) What are all of X's permissions in the current policy domain, and (v) Under what conditions does X have the permission to perform Y on resource Z. While answering these queries, the Rei engine takes into account all applicable policies and speech acts and tries to resolve any conflicts detected using the defined meta policies.

The engine is developed in Flora[2], an F-logic extension of XSB, and includes F-OWL [39], a reasoner for RDF and OWL. This allows policies to be described over ontologies in both RDF and OWL. The Rei engine has a Java wrapper for integration into different domains. The engine is modular and easy to extend with new deontic concepts (e.g. claims, privileges) and speech acts (e.g. promise, command).

As the policy engine is developed in Prolog, it is also possible to ask interesting queries through the prolog interface like, (i) Does John have the right perform any action that causes the room to become brighter, (ii) Can a student perform any action on a faculty printer, and (iii) Can Marty access any document that is of high priority. However, the Java interface for Rei does not currently support these kinds of queries.

In order to use Rei policies within an environment, the engine is usually queried by the enforcement mechanism before allowing/denying actions in the environment and for obligation management or used by an autonomous entity to decide what it should do next. However, having a policy described in Rei does not require the use of the Rei policy engine to enforce it. In some cases it is possible to compile such policies into another form that can more easily be enforced by a simple procedural mechanism [32].

2.2.2 Policy Analysis

To enable the development of consistent and valid policies, Rei provides two specifications: use-case management (also known as test-case analysis) and what-if analysis (also known as regression testing). The policy engine supports these two forms of analysis on policies and ontologies via a Java API that can be executed by policy engineers. In use-case management, the policy maker can specify a set of use cases that are verified against a set of policies. For example, the*Dean of the CS department should always have the permission to print to the HPColorPrinter.* This is defined as a use case and can be tested against the CS policy, the Lait lab policy and the school policy. If the policies and ontologies are consistent, the use case will be true. If the use case is false, the policy maker is aware that there is an error in specifications of the policies. On the other hand, what-if analysis is used to help the policy maker decide what changes to make to the policy or ontology. For example, *if I delete Rule1, will John still have the permission to print ?* or *if I add 'thesis-topic' property of value AI to George, will he still be prohibited from using the CS fax machine* are some examples of what-if analysis a policy maker may describe.

```
<analysis:WhatIfIRemoveRule rdf:ID="RemovingRule1">
     <analysis:policy rdf:resource="&dept;CSPolicy"/>
     <analysis:granting rdf:resource="&dept;Rule1"/>
</analysis:WhatIfIRemovePolicyRule>

<analysis:PermissionUseCase rdf:ID="CanJohnPrint">
     <analysis:actor rdf:resource="&inst;John"/>
     <analysis:action rdf:resource="&inst;Print"/>
     <analysis:target rdf:resource="&inst;LabPrinter"/>
</analysis:PermissionUseCase>
```

3. CASE STUDIES

Though Rei has been used in several research applications for policy management, we describe two that are most relevant to managing the behavior of entities on the web, (i) security, privacy, and confidentiality for semantic web services, and (ii) web privacy.

3.1 Security, Privacy, and Confidentiality for Semantic Web Services

Rei is used to define authorization, privacy and confidentiality policies for entities (both requesters and services) in the *Semantic Web Services* infrastructure. Based on our earlier work [6], a property called *policyEnforced* is defined as a subproperty of serviceParameter of OWL-S profile. This policyEnforced property is used to describe the different policies that have to be enforced for the correct execution of service. We use the same property for requesters, which could be agents, services or humans.

Access to services can be restricted through the use of *authorization policies.* These policies are made up of *permissions and prohibitions over attributes of the requester, service and the context at the time of invocation.* For example, an authorization policy of a printer web service could state that only students have the permission to access it.These policies constrain access to the service to only certain clients and under certain circumstances.

Assume the BravoAirService[3] as described on the OWL-S Web site has the following authorization policy *Only requesters who are in the same project as John or known to Tim have the right to access me if they fill out the comments.* We describe it in Rei as a permission over accessing BravAirService and define appropriate constraints.

```
<!-- some variables used for building the constraints -->
<entity:Variable rdf:ID="RequesterVar"/>
<entity:Variable rdf:ID="ProjectVar"/>

<!-- Get John's project -->
<constraint:SimpleConstraint rdf:ID="GetJohnProject"
     constraint:subject="&john;John"
     constraint:predicate="&foaf;currentProject"
     constraint:object="#ProjectVar"/>

<!--Is requester in the same project as John -->
<constraint:SimpleConstraint rdf:ID="SameProjectAsJohn"
     constraint:subject="#RequesterVar"
     constraint:predicate="&foaf;currentProject"
     constraint:object="#ProjectVar"/>

<!-- constraints combined -->
<constraint:And rdf:ID="AndCondition1"
```

[2]http://flora.sourceforge.net

[3]http://www.daml.org/services/owl-s/1.0/BravoAirService.owl

```
     constraint:first="#GetJohnProject"
     constraint:second="#SameProjectAsJohn"/>

<!-- Is requester known to Tim -->
<constraint:SimpleConstraint rdf:ID="IsKnownToTim"
     constraint:subject="&tim;Tim"
     constraint:predicate="&foaf;knows"
     constraint:object="#RequesterVar"/>

<!-- constraints combined -->
<constraint:Or rdf:ID="OrCondition1"
     constraint:first="#AndCondition1"
     constraint:second="#IsKnownToTim"/>

<!-- obligation to fill in the comments form -->
<deontic:Obligation rdf:ID="BravoProvision">
     <deontic:actor rdf:resource="#RequesterVar"/>
     <deontic:action rdf:resource="#FillComments"/>
</deontic:Obligation>

<!-- permission to use BravoService -->
<deontic:Permission rdf:ID="BravoPermission">
     <deontic:actor rdf:resource="#RequesterVar"/>
     <deontic:action
       rdf:resource="&bravo-service;BravoAir_ReservationAgent"/>
     <deontic:constraint rdf:resource="#OrCondition1"/>
     <deontic:provision rdf:resource="#BravoProvision"/>
</deontic:Permission>

<sws:AuthorizationPolicy rdf:ID="AuthPolicy">
     <policy:grants rdf:resource="#BravoPermission"/>
</sws:AuthorizationPolicy>

<!-- BravoAir-ReservationAgent enforces the AuthPolicy -->
<rdf:Description
      rdf:about="&bravo-service;BravoAir_ReservationAgent">
     <sws:policyEnforced rdf:resource="#AuthPolicy"/>
</rdf:Description>
```

In order to verify this policy and check whether a request is valid, FOAF descriptions of either John or Tim as well as that of the requester is required to be input to the Rei engine.

In this framework, we view *privacy policies as restricting access to those services that satisfy certain input/output conditions*. Information leakage is prevented through the use of *privacy policies*. For example, if a *user does not want her SSN number disclosed*, her privacy policy will restrict access to all those services that use SSN as output parameters.

Similarly it is possible to describe policies over input parameters of services. These are privacy policies associated with the requester. However, the privacy policies associated with a service describes how the information gathered by the service is utilized. This is similar to the P3P effort [5] and in our context we call these policies *privacy enforcement policies*. These policies are a set of obligations on the service regarding the usage of the information it collects from the requester. For example, *the privacy enforcement policy of a book selling Web site could be that it will not provide contact information of the requester to any third party and will only use it for statistical analysis*. This is translated into Rei as 'Obligation to not provide any contact information to third parties'. The requester could also define its privacy policy based on the privacy enforcement policy of the service. Consider a user whose privacy policy states that the user want to only use services that guarantee that they will not provide his contact information to third parties. In Rei, this privacy policy will be defined as 'Permission to access service that has an obligation to not provide contact information to third parties'.

Closely linked to privacy policies are *confidentiality policies that describe cryptographic characteristics of the input and output parameters of the service*. The problem of representing data confidentiality in the markup of semantic web services such as OWL-S is that encrypted data by its very nature does not reveal its internal value or structure because it is just a byte string. We therefore use a semantic markup that specifies the security characteristics of input and output parameters of web services while keeping information about the structure of the data without revealing its value and develop Rei policies over this markup. The markup includes the type of encryption/signing etc. and the base object. Consider as an example of a user who requires her SSN be encrypted. This means the user requires any service that needs her SSN as input parameters to use encryption. The user's policy will state that there the user is prohibited from using those web services whose input parameter type is SSN and is permitted to use those services whose input parameter type is 'encrypted' as described by the encryption ontology and whose base object is SSN.

We have developed an algorithm for testing policy compliance between requesters and services that can be integrated into the service selection process of the OWL-S MatchMaker [31]. This integration will allow the requester to invoke only those services that match the formers policies and whose policies are met by the requester.

3.2 Web Privacy

Though P3P is a very popular architecture, its deployment has been slow. We attribute it to two primary reasons. While one reason is the low adoption of P3P policy by websites, the other more direct reason is the inadequacy of APPEL. The limitations with APPEL include its notion of logical connectives, rule ordering and matching criteria. [1]. Due to these reasons APPEL can specify only what is unacceptable and not what is acceptable for a user. XPref, an XPath based language, tries to overcome some of these issues. Though XPref solves the issues with rule matching in APPEL, it does not solve the problem of restrictive expression capabilities of APPEL. Hence we propose the use of privacy policies described in an RDF based policy language, Rei, which can make policy decisions over actions and associated restrictions modeled using a Web privacy ontology. Rei not only allows expressive policies to be described over domain specific information but also provides explicit meta policies for rule ordering. These user policies can be evaluated over website policies published in P3P-RDFS[30] or over other website information. Through the use of Rei, a user's privacy preferences can be enforced even in the absence of a P3P policy of the website if the user specifies other constraints that the website fulfills.

As part of our attempt at improving client-side privacy protection, we have integrated the Rei policy engine into the P3P JRC proxy [4]. The proxy now reads P3P specifications of website and loads them into Rei if they are in RDF otherwise, using XSLT, converts them into RDF and then loads them into Rei. The Rei engine infers from the user's policy, the context information of the user, the trust information a user procures, and the P3P policy of a website, whether access to the website should be allowed or denied [26]. For example, *a user can define a Rei privacy policy over P3P RDF concepts and her context that states that access to a website that collects her clickstream data is only allowed if (i) she is in the*

[4]http://jrc.p3p.it

office and the policy is certified, (ii) the data is collected for improving the website and deleted after the current session, or (iii) the website is trusted by one of her colleagues. We describe this example using several well known ontologies including FOAF and P3P and a few of our own ontologies. The policy's *default behavior* is that everything that is not permitted is prohibited. We assume that the appropriate marked up information including the FOAF descriptions of the user and of her colleagues and the website description (that is either scraped off the website or obtained from a recommender network) are input to the Rei engine.

```
<!-- some variables used for building the constraints -->
<entity:Variable rdf:ID="WebsiteVar"/>
<entity:Variable rdf:ID="PolicyVar"/>
<entity:Variable rdf:ID="UserVar"/>
<entity:Variable rdf:ID="Someone"/>

<!--  privacy policy -->
<!-- everything that is not permitted is prohibited -->
<policy:Policy rdf:ID="PrivacyPolicy">
     <policy:grants rdf:resource="#Perm_AccessWebsite"/>
     <policy:defaultBehavior
       rdf:resource="&metapolicy;ExplicitPermImplicitProh"/>
</policy:Policy>

<deontic:Permission rdf:ID="Perm_AccessWebsite">
        <deontic:actor rdf:resource="#UserVar"/>
        <deontic:action rdf:resource="WP_Request"/>
      <deontic:constraint rdf:resource="#OrCondition2" />
</deontic:Permission>

<appel:Request rdf:ID="WP_Request">
     <action:actor rdf:resource="#User"/>
     <action:constraint rdf:resource="#DoesSiteCollectClickStream" />
</appel:Request>

<!-- Or condition -->
<constraint:And rdf:ID="OrCondition1"
     constraint:first="#IsUserInOfficeAndP3PCert"
     constraint:second="#NoRetentionAndTailoring"/>

<!-- Or condition -->
<constraint:And rdf:ID="OrCondition2"
     constraint:first="#OrCondition1"
     constraint:second="#IsWebsiteTrustedByColleague"/>

<!-- And constraint to check if website collects clickstream -->
<constraint:And rdf:ID="DoesSiteCollectClickStream"
     constraint:first="#P3PPolicy"
     constraint:second="#SiteCollectCS"/>

<!-- Is user in office and p3p policy is certified -->
<constraint:And rdf:ID="IsUserInOfficeAndP3PCert"
     constraint:first="#IsUserInOffice"
     constraint:second="#AndCondition3"/>

<!-- Get policy and check if its certified -->
<constraint:And rdf:ID="AndCondition3"
     constraint:first="#P3PPolicy"
     constraint:second="#IsPolicyCertified"/>

<!-- If no retention and purpose is tailoring -->
<constraint:And rdf:ID="NoRetentionAndTailoring"
     constraint:first="#AndCondition1"
     constraint:second="#IsNoRetention"/>

<constraint:And rdf:ID="AndCondition1"
     constraint:first="#DoesSiteCollectClickStream"
     constraint:second="#IsPurposeForTailoring"/>

<!-- find colleague and check if he/she trusts the website -->
<constraint:And rdf:ID="IsWebsiteTrustedByColleague"
     constraint:first="#IsColleague"
     constraint:second="#IsWebsiteTrusted"/>

<!-- Get the website's P3P policy -->
<constraint:SimpleConstraint rdf:ID="P3PPolicy"
  constraint:object=
  "http://www.w3.org/TR/p3p-rdfschema/p3p-rdf-schema.xml#policy"
  constraint:predicate="http://www.w3.org/2002/01/P3Pv1statement"
  constraint:subject="#PolicyVar" />

<!-- Does the website collect clickstream data -->
<constraint:SimpleConstraint rdf:ID="SiteCollectCS"
     constraint:subject="#PolicyVar"
     constraint:predicate="&p3p;data"
     constraint:object="&p3p;dynamic.clickstream"/>

<!--Is user in the office -->
<constraint:SimpleConstraint rdf:ID="IsUserInOffice"
     constraint:subject="#UserVar"
     constraint:predicate="&user;location"
     constraint:object="&user;AtWork"/>

<!-- Is the policy certified -->
<constraint:SimpleConstraint rdf:about="IsPolicyCertified"
     constraint:subject="#PolicyVar"
     constraint:predicate="&p3p;disputes"
     constraint:object="http://www.truste.org" />

<!-- Is the purpose for tailoring the website -->
<constraint:SimpleConstraint rdf:ID="IsPurposeForTailoring"
     constraint:subject="#PolicyVar"
     constraint:predicate="&p3p;purpose"
     constraint:object="&p3p;Purpose-tailoring" />

<!-- What is the retention period -->
<constraint:SimpleConstraint rdf:ID="IsNoRetention"
     constraint:subject="#PolicyVar"
     constraint:predicate="&p3p;usage"
     constraint:object="&p3p;Retention-no-retention" />

<!-- Is colleague  -->
<constraint:SimpleConstraint rdf:ID="IsColleague"
     constraint:subject="#UserVar"
     constraint:predicate="&foaf;knows"
     constraint:object="#Someone" />

<!-- Is website trusted -->
<constraint:SimpleConstraint rdf:ID="IsWebsiteTrusted"
     constraint:subject="#Someone"
     constraint:predicate="&user;trusts"
     constraint:object="#WebsiteVar" />
```

4. RELATED WORK

Role Based Access Control (RBAC) [9, 13, 33, 38] is one of the better known methods for access control, in which 'relations are established between users - roles, and permissions - roles'. It is difficult, however, to apply the RBAC model for systems in which it is not possible to assign roles to all users in advance. In addition, it is typically not possible to change access rights of a particular entity without modifying the roles. More sophisticated RBAC models include delegation between roles [2] and allow role assignments to users who are outside the system [14, 15]. The denomination for role-based delegation is the entire set of access rights associated with any one role that the delegator is in. In our work, however, the policy dictates what subset of permissions a delegator is allowed to delegate, to whom and under what conditions. Rei can actually be used to completely model RBAC. As of now, RBAC does not include an ontology language representation and is not easily extensible to different kinds of domain knowledge.

RBAC associates entities with roles and roles with permissions and so it does not take into account any other attributes of the entities or the environment in general.

Though they are both elegant solutions, PolicyMaker [28, 29] and KeyNote [3] work best in certificate-based systems and are not easily extensible. Delegation is controlled by a delegation depth and simple conditions on delegation. On the other hand, in Rei the permission to delegate is a separate permission and includes constraints not only on who can delegate, but whom they can delegate to. As Rei is capable of reasoning over domain knowledge in ontology languages, it is possible to develop constraints over attributes of requesters, actions, and the environment at different levels of abstraction. This is not possible in either PolicyMaker or KeyNote as they use a programming language for describing assertions. Rei is also capable of modeling both authorization and obligation policies, whereas KeyNote and PolicyMaker can only be used for authorization.

Extensible Access Control Markup Language (XACML) [10] is a language in XML for expressing access control policies. This work is similar to ours in that it allows control over actions and supports resolution of conflicts. On the other hand, it does not utilize, and thus does not benefit from, the interoperability and extensibility provided by semantic web ontology languages like RDF and OWL. It also does not model speech acts. Furthermore, its handling of conflict resolution across policies is more limited in expressiveness than Rei's.

There have been some efforts in adapting data exchange formats and protocols related to security in distributed systems to the Semantic Web like XML digital signatures [8], XML encryption [7], and X.509 Public Key Certificates [17]. These systems, based on XML [37], tend to be for authentication and accountability rather than authorization that our framework addresses. Though XML is probably the best exchange format for the Web, it only provides limited interoperability and scalability. Nevertheless, as these XML-based frameworks address issues different from our system, it is possible to use them in parallel with ours.

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. These 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 nor do they allow for expressive policies. 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 ours is a possible solution. Our work is not intended to replace the existing standardization efforts, rather we propose it as a semantic layer on top of the syntactically oriented standardization efforts. Future work will investigate algorithms that map our high-level descriptions into selected XML-based techniques.

KAoS is a policy language based in OWL [35]. It is similar to Rei and can be used to develop positive and negative authorization and obligation policies over actions. There are both advantages and disadvantages of the KAoS approach. 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. Using OWL, however, allows the classification of policy statements, enabling conflicts to be discovered from the rules themselves. On the other hand, the Rei engine can only discover conflicts (and resolve them using meta-policies) with respect to a particular situation and cannot pre-determine them statically. Another advantage that KAoS has is that if policy descriptions stay within OWL-Lite or OWL-DL, then the computation is decidable and has well understood complexity results. Also, KAoS includes mechanisms to ease enforcement, whereas the Rei system assumes that enforcement is handled separately from the policy engine. In terms of speech acts, however, KAoS only models simple delegations, whereas, Rei includes an integrated approach to speech acts for dynamic policy management, which is useful in autonomous distributed systems. Rei also provides specifications and tools for policy analysis and consistency checking that KAoS does not.

5. DISCUSSION

In this section, we discuss various aspects of Rei including why it uses a web ontology language, its usability, its computational complexity, and its relationship to OWL and SWRL.

5.1 Why Semantics ?

The web spans several domains, each of which use heterogenous domain-specific information. The policy management system used for governing the behavior of entities across the web should be easily extensible to be usable for any domain-specific information. As Rei uses OWL, it allows policies to be defined over different domain knowledge described in several ontology languages, without any change. The presence of semi-autonomous entities such as agents and web services, require that the policy management not only be understandable by humans, but should also be more automated, allowing these semi-autonomous entities to understand and use them appropriately. Rei is based on OWL that allows for greater machine understandability.

5.2 Usability

Rei has been used in several prototype research applications, both at UMBC and other research organizations, and has proved to be useful in a wide range of domains including semantic web services, pervasive computing, supply chain management systems, and multi-agent systems. The Rei policy engine is usually used as a querying engine and is encapsulated within another component that provides appropriate integration into the domain. For example, in the semantic web services domain, the Rei policy engine is integrated into the Matchmaker, which provides brokering between clients and web services. The domain policies and ontologies are loaded into the Rei engine via the encapsulating component at appropriate times. For example, during service discovery, the Matchmaker loads the policies associated with the service and client into the Rei engine and asks whether they are compatible. Depending on the application, the component queries the Rei engine about the permissions and obligations of entities in the system and uses these answers to manage their behavior.

As Rei does not restrict the domain knowledge, the communication protocols, or the manner in which the Rei engine can be used, it can be used in different domains to provide policy management for a wide range of applications.

5.3 Formal Semantics and Computational Complexity

Due to the complexity of the language and our focus on implementation and deploying applications, we have only just started looking at formal semantics and computational complexity for Rei. We intend to develop semantics largely in terms of courteous logic programs [11, 12]. In this section, we informally discuss some aspects of formal semantics and computational complexity.

Rei rules permit negation in the body and the form of negation is negation-as-failure (NAF). For the semantics of Rei, we adopt the well founded semantics (WFS) [36] for NAF as WFS always has a unique set of sanctioned conclusions, which is tractably computable (worst-case quadratic complexity) for the propositional case. We also make the assumption that the appearance of NAF in the rules satisfies the expressive restriction of *dynamic stratifiability*. Another aspect is the semantics of Rei's interaction with OWL reasoning over the domain ontologies. Rei treats OWL (i.e., Description Logic) inferencing essentially as an oracle and treats inferences from OWL axioms as virtual fact base. The body of a Rei rule in general is a boolean combination of triples. This is similar to the Lloyd-Topor extension of declarative logic programs, which is known to be tractably reducible to the ordinary case of rules (a.k.a. normal or ordinary logic programs) in which the body is a conjunction of atoms and/or negated atom [18]. Rei rules themselves do not result in new conclusions being drawn about the domain ontologies' predicates (classes and properties) as only a special set of predicates can appear in the head. Rei rules, like OWL, satisfy the Datalog (a.k.a. function-free) expressive restriction, i.e., there are no logical functions of more than zero arity. When ordinary or courteous logic program rules obey the Datalog restriction, together with the VB (number of distinct logical variables per rule is bounded) restriction, then the set of conclusions can be computed tractably; their qualitative computational complexity is similar to that of relational algebra and thus relational databases. We conjecture that Rei rules inferencing, modulo the OWL oracle/inferences, has similarly *tractable* computational complexity, perhaps under some additional relevant expressive restrictions.

5.4 Relationship between Rei and OWL

Though OWL-Full is not able to represent role-value maps such as 'uncle of' and 'same group as', it is able to describe a subset of Rei policies whose constraints involve checking the presence of common properties. A policy that states that all graduate students who are married are permitted to apply for on-campus housing can be represented in OWL-Full. This is possible by adding restrictions to properties of classes and by using classes as individuals and properties. For example, the above permission is described using an OWL-Full policy instance that has its actor property restricted to the class of 'MarriedGraduateStudent' and its action property restricted to the class of 'ApplyForOnCampusHousing' actions. 'MarriedGraduateStudent' is defined by restricting the 'marital status' property of graduate students to 'married'. In Rei, this permission would be defined by a boolean combination of constraints that would check whether the actor was of type 'GraduateStudent', if the actor had a property 'marital status' that was set to 'married', and if the action was of type 'ApplyForOnCampusHousing'. However, if contextual information is required, then a new property must be added. For example, to restrict the previous permission to during school hours, a new property called context is added to the action class and a restriction over context is defined. Using OWL-Full in this way increases the complexity of the policy and computation becomes undecidable. Also, there are very few reasoners are able to correctly interpret all the test cases associated with OWL-Full.

Rei uses OWL-Lite to describe its essential classes, properties, and relationships. The Rei policy engine reasons over these classes and properties to support the description of positive and negative rule-based policies. OWL-Lite is decidable and has lower formal complexity than OWL-DL and OWL-Full. The OWL-Lite constructs used in Rei are supported by most existing reasoners. Though we use variables to describe constraints, the reasoning required is provided by XSB and the computability is tractable. We believe it is easier and more intuitive to develop constraints as defined by Rei (as boolean combinations of Subject, Predicate and Object) than to create OWL-Full instances using restricted classes. Rei also supports the definition of relations between classes that give it greater expressivity than OWL.

SWRL is a a rule language based on OWL-Lite, OWL-DL, and RuleML [16]. SWRL includes a high-level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sublanguages of OWL. Rei uses OWL-Lite to describe its classes and properties and also allows variables to be used for developing constraints that are not possible in RDF, DAML+OIL or OIL. This adhoc extension of OWL is able to define constraints associated with instances, whereas SWRL is a language for defining rules. We are considering redoing Rei in SWRL in a few months when SWRL and some supporting tools are more mature. This will probably be an improvement and will strengthen Rei because we will not be introducing our own way of encoding rules.

6. SUMMARY

Though the web is already a vast network of shareable information, in order to realize its full potential, regulating the behavior of web entities is essential. We propose that declarative policies be used to describe the behavior of web entities and web protocols be modified for policy exchange, negotiation, compliance checking, and for checking whether the specified policies were fulfilled. Rei attempts to meet some of the requirements imposed by the characteristics of the web by (i) including a machine-understandable specification language, a policy engine analysis tools, (ii) allowing policies to be described in terms of attributes of users, actions, and other context and supporting meta-policies for conflict resolution, (iii) providing greater extensibility as policies can be described over domain knowledge at different levels of abstractions, and (iv) supporting the dynamic policy modification via speech acts. While Rei policies can be used to describe different kinds of behavior of web entities, the policy engine can be used for policy compliance, fulfillment, and enforcement. We've developed two research applications that explore the utility of Rei for policy management for the web, namely web privacy and security and privacy for web services. Though, there still is work to be done in order to provide a

complete policy management framework for the web, we believe Rei has proved to be a good starting point.

7. ACKNOWLEDGEMENTS

The authors would like to thank Dr. Benjamin Grosof for his help on the formal semantics and computational complexity of Rei.

8. REFERENCES

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. An XPath-based preference language for P3P. In *Proceedings of the Twelfth International Conference on World Wide Web*, pages 629–639. ACM Press, 2003.

[2] E. Barka and R. Sandhu. Framework for Role-Based Delegation Models. In *Annual Computer Security Applications Conference*, 2000.

[3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System Version. Internet RFC 2704, September 1999., 1999.

[4] M. Cornwell, J. Just, L. Kagal, and T. Finin. A Policy Based Collaboration Infrastructure for P2P Networking. In *Twelfth International Conference on Telecommunication Systems, Modeling and Analysis, July 2004*, 2004.

[5] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. Platform for Privacy Preferences (P3P). http://www.w3.org/P3P, 2002.

[6] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for DAML Web Services: Annotation and Matchmaking. In *Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003*, 2003.

[7] D. Eastlake and J. Reagle. XML Encryption Syntax and Processing. W3C Candidate Recommendation, August 2002.

[8] D. Eastlake, J. Reagle, and D. Solo. XML-Signature Syntax and Processing. RFC 3275, March 2002.

[9] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

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

[11] B. Grosof. Courteous Logic Programs: Prioritized Conflict Handling for Rules. IBM Research Report RC20836, May 1997, Revised December 1997.

[12] B. N. Grosof. A Courteous Compiler From Generalized Courteous Logic Programs To Ordinary Logic Programs. Report included as part of documentation in the IBM CommonRules 1.0 alpha prototype Web release of July 30, 1999 on AlphaWorks, July 20 1999.

[13] L. Guiri. A New Model for Role-based Access Control. In *11th Annual Computer Security Application Conference, pages 249–255, New Orleans, LA, December 11-15*, 1995.

[14] A. Herzberg, Y. Mass, J.Mihaeli, D.Naor, and Y. Ravid. Access Control meets Public Key Infrastructure : Or Assigning roles to strangers. In *2000 IEEE Symposium on Security and Privacy, Oakland, May 2000*, 2000.

[15] T. Hildmann and J. Barholdt. Managing trust between collaborating companies using outsourced role based access control. In *Fourth ACM workshop on Role-based access control, Fairfax, Virginia, United States*, 1999.

[16] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: Semantic Web Rule Language Combining OWL and RuleML. Version 0.6 of 30 April 2004. http://www.daml.org/rules/proposal/, 2004.

[17] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280, April 2002.

[18] J. W. Lloyd. *Foundations of logic programming; (2nd extended ed.)*. Springer-Verlag New York, Inc., 1987.

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

[20] L. Kagal and T. Finin. Modeling Conversation Policies using Permissions and Obligations. In *AAMAS 2004 Workshop on Agent Communication (AC2004)*, July 2004.

[21] L. Kagal, T. Finin, and A. Joshi. Trust Based Security for Pervasive Computing Enviroments. In *IEEE Communications, December 2001*, 2001.

[22] L. Kagal, T. Finin, and A. Joshi. A Policy Based Approach to Security for the Semantic Web. In *Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003*, 2003.

[23] L. Kagal, T. Finin, and Y. Peng. A Delegation Based Model for Distributed Trust. In *Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents, International Joint Conferences on Artificial Intelligence*, 2001.

[24] L. Kagal, T. Finin, and Y. Peng. A Framework for Distributed Trust Management. In *IJCAI-01 Workshop on Autonomy, Delegation and Control*, 2001.

[25] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, and K. Sycara. Authorization and Privacy in Semantic Web Services. In *IEEE Intelligent Systems (Special Issue on Semantic Web Services)*, 2004.

[26] P. Kolari, L. Kagal, A. Joshi, and T. Finin. Enhacing P3P Framework with Policies and Trust. UMBC Technical Report and under review, 2004.

[27] R. Masuoka, B. Parsia, and Y. Labrou. Task Computing - the Semantic Web meets Pervasive Computing. In *2nd International Semantic Web Conference (ISWC2003)*, 2003.

[28] M.Blaze, J.Feigenbaum, and J.Lacy. Decentralized Trust Management. *Proceedings of IEEE Conference on Privacy and Security*, 1996.

[29] M.Blaze, J.Feigenbaum, and M.Stauss. Compliance Checking in the Policy Maker Trust Management System. In *Proceedings of Financial Crypto'98, Lecture Notes in Computer Sciences vol.1465, Springer Berlin, 1998*, 1998.

[30] B. McBride, R. Wenning, and L. Cranor. An RDF Schema for P3P. W3C Note 25 January 2002.

[31] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *ISWC2002*, 2002.

[32] A. Patwardhan, V. Korolev, L. Kagal, and A. Joshi. Enforcing policies in Pervasive Environments. In *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, 2004.

[33] R. S. Sandhu. Role-based access control. In M. Zerkowitz, editor, *Advances in Computers*, volume 48. Academic Press, 1998.

[34] J. Undercoffer, F. Perich, A. Cedilnik, L. Kagal, A. Joshi, and T. 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.

[35] 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. In *International Semantic Web Conference (ISWC 03). Sanibel Island, Florida*, 2003.

[36] A. van Gelder, K. Ross, and J. S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.

[37] W3C. Extensible Markup Language. W3C Recommendation, http://www.w3c.org/XML/.

[38] N. Yialelis, E. Lupu, and M. Sloman. Role-based Security for Distributed Object Systems. In *Fifth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '96) (1996), pp. 80-85*, 1996.

[39] Y. Zou, H. Chan, and T. Finin. F-OWL: an Inference Engine for Semantic Web. In *Third NASA-Goddard/IEEE Workshop on Formal Approaches to Agent-Based Systems (FAABS III), 26-28 April 2004, Greenbelt MD*, 2004.