A Policy Based Approach to Security for the Semantic Web[*]

Lalana Kagal, Tim Finin, and Anupam Joshi

Computer Science and Electrical Engineering Department
University of Maryland Baltimore County
Baltimore, Maryland, USA
{lkagal1,finin,joshi}@cs.umbc.edu

Abstract. Along with developing specifications for the description of meta-data and the extraction of information for the Semantic Web, it is important to maximize security in this environment, which is fundamentally dynamic, open and devoid of many of the clues human societies have relied on for security assessment. Our research investigates the marking up of web entities with a semantic policy language and the use of distributed policy management as an alternative to traditional authentication and access control schemes. The policy language allows policies to be described in terms of deontic concepts and models speech acts, which allows the dynamic modification of existing policies, decentralized security control and less exhaustive policies. We present a security framework, based on this policy language, which addresses security issues for web resources, agents and services in the Semantic Web.

1 Introduction

The Semantic Web is a future generation of the current Web, where resources are annotated with machine understandable meta-data, allowing the automation of the retrieval and usage of these resources in their correct contexts. The focus of much of the existing semantic web work [1, 2] has been on the creation of languages for knowledge representation such as DAML+OIL (`http://www.daml.org/`) and OWL (`http://www.w3.org/TR/webont-req/`), of systems that reason over these semantic descriptions for information retrieval [3, 4], and tools to help annotate web pages with semantic markup [5, 6]. However, with the increase in interest in web based e-commerce, banking and travel services, the amount of business that is transacted online, and the explosion in the amount of services available, a key need for the vision of the Semantic Web to succeed is the ability to handle security and privacy and the ability to automate these protocols for the use of all web entities alike. The obvious solution is to extend security mechanisms applicable to distributed systems (e.g. Kerberos, PGP, SPKI etc.) for the semantic web. However, we believe that this is difficult given

[*] This research was supported in part by Hewlett-Packard Labs, Palo Alto, DARPA contract F30602-97-1-0215, NSF career award 9875433 and NSF award 0242403.

the completely decentralized nature of the web, the extremely large number of resources, services, agents and users, and their heterogeneity. In addition, the set of entities that access an information source or interact with a given web entity (a web entity includes web pages, agents and services) cannot be enumerated a priori. Along with this is the problem of the semantic meaning of the security information; it is not feasible to expect all entities to use the same terminology to represent security protocols and information. This forces the use of a semantic language like RDF-S, DAML+OIL or OWL whose constructs help entities better understand the meaning of the security information. Based on this, we conclude that a security framework for the Semantic Web needs to be flexible, semantically rich and simple enough to automate.

We propose the use of two main components in order to secure the Semantic Web; a semantic policy language for defining security requirements and a distributed policy management approach. As the Semantic Web is composed of web entities like web services, agents, and human users, security should be uniformly applied to and be applicable to all. It is important for web entities to be able to express their security information in a clear and concise manner so that its meaning is unambiguous. Towards this end we suggest the use of a policy language, based on RDF-S, to markup security information. Though, DAML+OIL and OWL include better semantics, they are still evolving and as part of our future work, we are working towards developing a mapping from our policy language into one of them. In distributed policy management, every entity in the system can specify and enforce its own policy as there is no central policy. Policies can be specified in terms of properties of users, agents, services and resources and not just their identities as it may not always be possible to authenticate all entities on the web. The framework attempts to overcome several deficiencies of current security mechanisms by providing access control to resources from a large number of requesting entities that may be unknown to the former, providing security without necessarily authenticating requesters completely, providing flexibility in specifying security requirements and giving every entity a certain amount of autonomy in making their own security decisions. Given the inherently distributed nature of the semantic web, our framework supports several speech acts that allow entities to automatically update existing policies : the delegations of permissions, by which web entities can delegate to other entities certain rights that they possess bound by constraints, the revocation of permissions to remove existing rights, the requesting of actions to be performed and rights to be delegated and the cancellation of previous requests.

In Section 2 we describe some related security work and in Section 3 we discuss the overview of our approach. The details of the policy language, including the constructs, conflict resolution and speech acts supported, follow in Section 4. As delegation is an important aspect of our framework, in Section 5 delegation management is discussed. We discuss the details of our framework with respect to different web entities in Section 6 and we conclude with a summary.

2 Existing Security Mechanisms

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 [7], XML encryption [8], and X.509 Public Key Certificates [9]. These systems, based on XML [10], 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. Semantic languages like RDF-S, DAML+OIL or OWL that describe ontologies and provide interoperability across applications are more suitable for the Semantic Web. However, as these XML-based frameworks address issues different from our system, it is possible to use them in parallel with ours.

Extensible Access Control Markup Language (XACML) [11] 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, (i) it does not provide support for speech acts, (ii) adding domain specific information is time consuming (In our framework, adding application/domain specific information involves importing those ontologies or extending our policy ontology.), (iii) it does not allow *types* of actions to be defined (Our policy language allows policies to be written for types of actions that satisfy a certain set of conditions.), (iv) its conflict resolution is not across policies, (v) it can only be used for authorization policies (Our policy language can be used for policies about authorization, privacy, conversation etc.), and (vi) as it is based in XML, it does not benefit from the interoperability and extensibility provided by a semantic language.

The Platform for Privacy Preferences (P3P) is a standard developed by the World Wide Web Consortium (W3C) that enables websites to describe their privacy policies and allows browsers to reason over these policies to decide whether they match the user's preferences [12]. This work tries to reduce Internet users' concerns about the personal information they reveal when visiting various websites. A website specifies its privacy policy and a user is not allowed access to it or warned of conflict if his preferences do not match the policy. The specifications are not designed for information privacy but for the information gathered by websites, and they cannot be used to describe privacy policy for agents and services on the web.

Role Based Access Control (RBAC)[13–16] is one of the better known methods for access control, in which relations are established between users - roles, and permissions - roles. However it is difficult 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. Simple Role Based Access Control is rather restrictive, as every time access rights of an agent change, its role has to change as well. Our policy language allows access rights (and the other deontic constructs) to be associated with different credentials and properties of entities, and not roles alone. More sophisticated RBAC models include work on delegation between roles [17] and assigning roles to users that are outside the system [18, 19]. This research considers delegating the entire set of permissions associated with a set of roles of the delegator (the entity doing the delegation) to the delegatee (the entity receiving the delegation). Our work allows the the policy to dictate what subset of permissions a delegator is able to delegate, to whom and under what conditions. In fact, our policy language is able to represent most RBAC models. Herzberg [18] uses properties of certificates to map users outside the domain to domain specific roles. This is very similar to our work, however instead of mapping properties of certificates to roles, we map properties of the user to access rights directly. Though this may seem like a drawback, we have found that this provides greater control in specifying access rights to foreign users as certain *types* of rights can be given, like the read rights on a certain resource, the right to print to all the color printers on the fifth floor, and the right to delete all the files belonging to your colleague.

The Strongman project uses KeyNote [20] to build secure information systems by automating key and policy management [21]. KeyNote is a trust management system, which binds public keys to access rights. It does not verify certificates or signatures, but allows the functionality to be handled by an external program making integration with existing PKI infrastructures possible. The system is versatile but rather too closely bound to digital certificates as it cannot handle other domain information or link into non PKI systems. It includes a rather simple notion of delegation and uses a programming language instead of a semantic language making its integration with the semantic web difficult.

3 Our Approach

Drawing from our experience with security issues associated with widely distributed systems [22–24], we propose a security framework for the Semantic Web based on distributed policy management through the use of a semantic policy language. By distributed policy management, we mean every web entity can specify policies for its access, for privacy, for entities it wants to communicate with etc., which are enforced either by an internal policy engine or the policy engine on the platform it is running on or registered with. A policy includes a set of rules that associate a required set of credentials with a certain ability or right; implying that only entities with the specified credentials can possess the ability. A credential is any property associated with a entity: affiliation, age, certificate, login/password, membership in a certain organization, etc. An entity that meets the requirements of a policy rule is trusted to use the associated action. The policy language, based on deontic concepts, consists of constructs that are flexible and that allow different kinds of policies (security, privacy, management, conversation etc.) to be specified. The language is not tied to any specific application and permits domain specific information to be reasoned over. As the framework is geared towards environments that consist of several domains the language includes meta-policies for specifying mechanisms for handling conflicts.

Though policies associate credentials with actions or services, the process of verifying credentials is handled outside the system. This allows existing mechanisms like PKI, Kerberos etc. to be integrated into the framework.

Our work is similar to role based access control - in that a user's access rights are computed from his/her properties. However, we propose to use additional ontologies that include not just a fixed property, role, but any properties and constraints expressed in an semantic language including elements of both description logics and declarative rules. Consider the earlier example of an employee delegating his *speaker* role to a visiting speaker. In our work, the employee is able to delegate just his rights on the projector and coffee maker to the speaker instead of all the rights associated with the speaker role. This provides greater control on delegation. It also allows rights to be assigned dynamically to a person, who has no role within that organization, without creating a new short term role specific to this person. Similarly, rights can be revoked from a user without changing his/her role, making this approach more flexible and maintainable than role based access control.

The main points of our research work include

- Semantic Policy Language : Our framework provides a policy language based on deontic logic concepts for specifying entities, properties of entities, actions, speech acts, policies. These policies are based on properties of agents, and other domain specific conditions and are described in terms of permissions (what an agent can do), prohibitions (what an agent cannot do), obligations (what an agent should do), and dispensations (what an agent need no longer do). Each web entity is able to describe its security policies in an intelligible manner, so that the meaning is distinct. This allows all entities interacting with it to unmistakably understand its security requirements. The language has an ontology that allows policies (including meta policies) to be described in RDF-S [25], though policies in Prolog [26] are also accepted. Using the axiomatic semantics of RDF-S and the policy language, interacting entities are able to interpret the security information correctly.
- Modeling Speech Acts : We provide a specification of speech acts that can be used to extend the policies namely; delegation (add a permission), revocation (remove a permission or add a prohibition), request (causes an action to be performed or causes a delegation which leads to a permission) and cancel (cancels previous request). Our framework includes detailed delegation management that allows access rights to be transferred further decentralizing control and allowing policies to be less comprehensive. The inclusion of speech acts also allows conversation policies to be described, which is important for controlling communication.
- Distributed Security Framework : In most systems, policies are centralized, specified by single set of individuals, about a predetermined set of users and resources and in a single location with a central point of control. However the widely distributed, open systems that we are addressing require decentralized policies and distributed control, where every entity is responsible

for describing and enforcing its own policy, which our framework supports. We provide a security framework that uses the policy engine in a distributed way allowing a web entity flexibility in security support.

– Mechanisms for Conflict Resolution : Since our framework allows every entity to specify its own policy, and as the Semantic Web spans several domains there is a potential for conflicts. Towards this end, we provide constructs for resolving conflicts; namely setting priorities for rules or entire policies and specifying precedence between positive and negative modalities for policies grouped by agents or actions/services.

The following sections discuss the above mentioned points in greater detail.

4 Policy Language

Our policy language is modeled on deontic concepts of rights, prohibitions, obligations and dispensations [27]. We believe that most policies can be expressed as what an entity can/cannot do and what it should/should not do in terms of actions, services, conversations etc., making our language capable of describing a large variety of policies ranging from security policies to conversation and behavior policies. The policy language has some domain independent ontologies but will also require specific domain ontologies. The former includes concepts for permissions, obligations, actions, speech acts, operators etc. The latter is a set of ontologies, used by the entities in the system, which defines domain classes (person, file, deleteAFile, readBook) etc. and properties associated with the classes (age, num-pages, email).

The language includes two constructs for specifying meta-policies that are invoked to resolve conflicts; setting the modality preference (negative over positive or vice versa) or stating the priority between policies [28]. For example, it is possible to say that in case of conflict the Federal policy always overrides the State policy.

Another important aspect of the framework is that it models speech acts like delegation, revocation, request and cancel that allow policies to be less exhaustive and allow for decentralized security control. Delegations are very important to the internet because web 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 access rights of an entity to be propagated to a set of trusted entities, without explicitly changing its policy or requirements. Consider a DAML-S web service that is only accessible by users of a certain role in a certain organization. These users also have the right 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, while the constraints associated with the delegation are met. The ability to delegate makes it possible to change access rights of web entities dynamically. A revocation speech act nullifies an existing right (whether policy based or delegation based) of an entity. An entity can request another entity for a right or to perform an action on its behalf and an entity can also cancel any previously made request.

The policy language supports individual policies as well as group and role based policies in a uniform manner by allowing domain dependent representations for roles and/or groups to be included in the conditions of the policy rules.

The policy language includes a RDF-S ontology for specifying entities, objects, policies, credentials, and domain specific information.

4.1 Representation of Actions and Conditions

Though the actual execution of services/actions is outside the system, the policy language includes a representation of them that allows more contextual information to be captured and allows for greater understanding of the action and its parameters. This allows policies to be defined over different aspects of an action and not just its identity. For example, it is possible to say all faculty members of the CS department have the right to all services (e.g. print, fax, xerox) on any color printer in the UMBC CS department. Action specifications include the target resources they act on, pre-conditions and effects. Complex actions can be composed using action operators of *sequence, non-deterministic choice, once* and *iteration.*

Simple conditions are built from domain specific information and complex conditions can be built from simple conditions using *and, or* and *not.* A condition is of the form "entity from UMBC" or "in the role of student".

Using these actions and conditions, a policy maker is able to create policy rules. There are four kinds of policy rules representing each deontic concept : right, obligation, prohibition, and dispensation. Each policy rule is a tuple of action and associated requirements. In order to associate these policy objects with users, the policy maker uses the *has* predicate in logic or the *grants* property of the *Policy* class in RDF-S creating rights, obligations etc. for the users.

Example 1 : Simple condition. A web service specifies that all graduate students from University of Maryland Baltimore County can access its service, which is identified by *service1.*

The policy rule in logic is as follows :

```
has(X, right(service1, graduateStudent(X, 'UMBC')))
```

where, service1 is an allowable action, UMBC is a constant for University of Maryland Baltimore County and graduateStudent(X, 'UMBC') is a domain dependent condition specified by the deployer. It is assumed that either the matchmaker, which the service is registered with, or the service itself it able to verify the attributes of an entity using traditional schemes like PKI, SPKI etc.

We are experimenting with ways to encode our simple policy rules in RDF and Notation3 [29]. Eventually, we hope to use whatever standard or standards emerge from the semantic web community for representing rules in RDF. Here is our N3 example in one framework we are using. We describe the rest of our examples in Prolog for conciseness.

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rei: <http://www.csee.umbc.edu/~lkagal1/rei-simplified#>.
@prefix univ: <http://www.csee.umbc.edu/~lkagal1/univ#>.
@prefix : <#>.
:v1 a rei:Variable.
:R a rei:Right;
        rei:agent :v1;
        rei:action univ:service1.
:cseepolicy a rei:Policy;
rei:grants [a rei:granting;
        rei:to v1;
        rei:deontic R;
        rei:oncondition [a rdfs:statement;
                rdfs:subject v1;
                rdfs:predicate rdfs:type;
                rdfs:object univ:GraduateStudent]].
```

Example 2 : Complex Condition. Consider a right associated with 'John'. *'John'* has the right to either perform action *printBW* followed by repeated executions of *printColor* or perform action *fax* once. He only has the right while he satisfies the associated conditions.

```
has('John', right(nond(seq( printBW, iteration(printColor)),
once(fax)),lab-member('John','AI')))
```

4.2 Speech Acts

The language models four speech acts that allow policies to be less exhaustive, allow for decentralized security control and are able to modify policies dynamically: delegate, revoke, cancel and request.

- Delegation : A delegation from an entity gives a right to another entity or group of entities. A delegation, if valid, causes a *Right* to be created. Only an entity with the *Right* to delegate can make a valid delegation. A delegator always retains the right to revoke the delegated right. (Please refer to Section 5 for more details about delegations)
  - *delegateSpeechAct(Sender, Receiver, right(Action, Condition))* and Receiver satisfies Conditions $\longrightarrow$ *has(Receiver,right(Action, Condition))*
- Request : There are two kinds of requests; a request for an action and a request for a right. The former, if accepted, causes an obligation on the part of the receiver. A request for a right, if valid and accepted, causes the receiver to delegate the right to the sender. However, the delegation allows the sender the permission to perform the action only if the receiver has the right to delegate.

- *requestSpeechAct(Sender, Receiver, Action)* $\longrightarrow$ disagree
  requestSpeechAct(Sender, Receiver, Action) $\longrightarrow$ *has(Receiver, obligation(Action, Condition))*
- *requestSpeechAct(Sender, Receiver, right(Action, Condition))* $\longrightarrow$ disagree
  request(Sender, Receiver, right(Action, Condition)) $\longrightarrow$
  delegateSpeechAct(Receiver,Sender, right(Action, XCondition))

– Revoke : Revocation is the removal of a right and acts as a prohibition. A revocation is allowed in two cases; an entity can revoke those rights to which it has the right to revoke or those rights that it has itself delegated.
  revokeSpeechAct(Sender, Receiver, right(Action, Condition)) $\longrightarrow$ *revocation(Receiver, right(Action, Condition)) and Sender no longer has the right to perform Action*

– Cancel : An entity can cancel any request it has sent, and this nullifies any obligation or delegation caused by the request.
  - *cancelSpeechAct(Sender, Receiver, Action) AND has(Receiver, obligation(Action, true))* $\longrightarrow$ *has(Receiver, dispensation(Action, Condition))*
  - *cancelSpeechAct(Sender, Receiver, right(Action, X))* $\longrightarrow$ *Sender no longer has the right to perform Action*

Example 3 : Request. Continuing with Example 2, 'Jane' is not a lab member of 'AI' and she requests John to give her the right to perform *printBW*.

```
requestSpeechAct('Jane', 'John', right(printBW,_))
```

If accepted and if 'John' has the right to delegate, 'John' sends a delegation to 'Jane' giving her the right to use printBW as long as she is an employee of 'UMBC'.

```
delegateSpeechAct('John', 'Jane', right(printBW,employee('Jane','UMBC'))
```

4.3 Meta Policies

As there is a potential for conflicts, the language also contains meta-policy specifications for conflict resolution. Meta-policies are policies about how policies are interpreted and how conflicts are resolved dynamically. Conflicts only occur if the policies are about the same action, on the same target but the modalities (right/prohibition, obligation/dispensation) are different. Meta policies in our system regulate conflicting policies in two ways by specifying priorities and precedence relations [30]. Using a special construct, *overrides*, the priorities between any two rules or two policies can be set. In case of conflicts, the rule/policy with the higher priority is chosen. It also possible to indicate the order in which the priorities are checked; policy level or rule level first. The other way of resolving conflicts is by specifying which modality holds precedence over the other in the meta-policies. The deployer of the platform/agent can associate a certain precedence for all policies associated with a set of actions or a set of agents satisfying a certain set of conditions.

4.4 Policy Engine

Associated with the policy language is a policy engine that interprets and reasons over the policies, related speech acts and domain information expressed either in Prolog or RDF-S to make decisions about applicable rights, prohibitions, obligations and dispensations. It is possible to extend the policy ontology with other ontologies, which will be accepted and reasoned over by the policy engine. All domain specific information can be specified as separate ontologies and domain specific actions can be subclasses of DomainAction of the policy ontology.

If, while processing a request, the policy engine comes across a conflict, it uses associated meta-policies to resolve it. For example, 'John' wants to print to 'HPPrinter021', he sends a request to 'HPPrinter021'. "HPPrinter021' finds that 'John' has both a right to use it and is prohibited from using it. However, the right has been given higher priority than the prohibition and 'HPPrinter021' permits 'John' to print. The policy engine is able to answer questions about whether a request is valid, what the pending obligations of an agent are, under what conditions a request for a certain action is authorized etc. In order to make correct policy decisions, we assume the presence of a monitoring service that sends all relevant speech acts to the policy engine. The policy engine is implemented in Prolog with a Java interface.

The engine is used as a reasoning engine and is consulted before every request for an action/service and before taking any action. Based on the answer of the engine, the controller of the web entity (could be the entity itself) decides whether to allow or deny the request or go ahead with the action. The engine is also capable of answering other queries related to policy making: who can perform a certain action, who can perform any action on a certain resource, what conditions must an entity possess in order to use a certain action/resource, what unfulfilled obligations does entity X currently have, etc. These queries can be used to decide whether the policy does what it is supposed to and to check whether there are any inconsistencies in it.

5 Delegation Management

Our framework allows entities to delegate rights, with restrictions attached, to other entities. A delegation is in fact a transference of a right from one entity (delegator) to another entity (delegatee). However after the delegation, the delegator retains the right. The right to delegate is defined implicitly and explicitly. Implicitly, an entity can delegate rights to any service it offers and explicitly, an entity that has been given the right to delegate by another entity, who has the right, can perform valid delegations, as long as the delegation fulfills the constraints of the previous delegation. A delegation usually has constraints attached, such as one that limits the access to a certain period, or to whom the right can be re-delegated. A delegation consists of various information; delegator, right, constraints on delegatee, constraints on execution, constraints on re-delegation and time period. By using constraints on delegatee, the delegator can specify whom to delegate to. For example, a delegation could be conferred on all agents with certificates from a certain CA and that are registered with a certain platform. By restricting which delegatee can actually use the right, the delegator can prevent wrongful execution of the right. Constraints on re-delegation allow the delegator to decide whether the right can be re-delegated and to whom it can be re-delegated to.

In a delegation chain, an entity has the right to a certain action (including speech acts) if it possesses the right or if it has been delegated the right. It should satisfy the conditions associated with the innermost right of execution of every delegation in the chain. Each delegator should satisfy the condition on the delegation of the delegation before it in the chain and the delegatee conditions of all previous delegations. If any one entity fails any required delegator condition (whether caused by a change in circumstances or a new revocation or the expiration of a delegation), the delegations from that point on in the chain are invalid. We have developed rules that capture this information and enforce security by checking these constraints for all entities in a certificate chain.

Example 4 : Delegation. 'Amy' has the right to delegate the right to execute printOnePageHP and she delegates this right to 'Tim'. 'Tim' can only execute printOnePageHP if he satisfies both the condition associated with the speech act (employee('Tim', 'UMBC') and the condition associated with 'Amy''s delegation right (group-member(X, Group)).

```
has('Amy', right('Amy',
     delegate(right(X, print, group-member(X,Group))), employee('Amy', 'UMBC')))

delegateSpeechAct('Amy', 'Tim', right('Tim', print, employee('Tim', 'UMBC')))
```

6 Design Overview

In this section, we present a security framework that realizes our previously described ideas about distributed policy management with the help of our policy language. We describe the security functionality of the three main categories of entities prevalent on the semantic web; web services, agents and web resources like pages, servers, etc. The framework is similar for all three cases. A web entity specifies its security policy and consults a policy engine before every request for action by other entities and before any action it performs. This policy engine could be internal i.e. part of the web entity itself or it could belong to a broker or platform that the entity is running off. If the policy engine is external, then the entity has to inform the external engine module of its policy (either explicitly or as part of its markup) and has to trust the broker/platform to filter requests to it based on its policy. An entity also specifies a privacy policy, which is consulted before the entity or the broker/platform communicates any information about the entity.

6.1 Web Resources

In the internet of today there are a lot of pages (static and dynamic) that require some form of authentication and access control before allowing requests to go through, e.g. accessing bank statements, submitting news items etc. This number will keep growing as the amount of sensitive information accessible over the net will keep increasing. To allow agents, services and humans alike to use these secure pages, they are marked up with our policy language. The markup specifies what credentials are required for access in terms of username/password, certificate, and other credentials [31]. The web server that is hosting the web resource contains a policy engine module. The web server is trusted to perform the security functionality on behalf of the resource, which holds even in the current web. When a secure web resource is requested, the server reads the page and checks the security markup. It feeds this information along with the parameters associated with the request to the policy engine module. The policy engine module reasons over the request, the credentials of the requester and the policy of the web resource to decide whether the request should be allowed to go through. If the credentials don't match, the module could return the list of credentials required for access, in a form understandable by the requester otherwise it returns the requested resource data.

This framework is based on an earlier policy based system we developed for eHealthcare in pervasive computing environments [32], where electronic patient information is redacted according to the requester's credentials (including his role in the hospital, certificates etc.), his relationship to the patient and any delegations he possesses, providing different views of the same information to different hospital personnel.

Example 5 : As an example, consider a hospital web site that provides patient information. The patient information web page requires that all requests be accompanied by valid certificates provided by the hospital. One of the doctors, who is away on leave, is discussing a difficult case with one of his friends, who is also a doctor. His friend makes a suggestion the doctor has not tried as yet. The doctor uses his cellphone to retrieve certain information from the hospital's website. His cellphone sends an HTTP request to the website. The web server rejects the request because there is no certificate. The doctor then asks his web agent (which is running on a server at his home) to retrieve the information. When the website denies the agent's request, the agent asks for the required list of credentials. The web server replies that it requires a hospital authorized certificate. The agent is able to understand this credential perfectly and resends the request with the necessary certificate attached. This time the web server permits the request and the agent returns the information to the requesting doctor.

6.2 Agents

Agents that search the web for information/services have to semi-autonomously decide what web services, resources and agents to trust about certain information and services. Along with this, each agent also has to decide what entities to trust to use its own services and information.

We currently follow the the FIPA specifications for agents [33]. Agents exist within agent platforms, which provide middleware functionality to all registered agents. These platforms provide registration, and communication services to the agents, along with enabling them to provide and access services. The two core facilitator agents are Agent Management System (AMS), and the Directory Facilitator (DF). The AMS maintains a directory of registered agents, containing their Agent Identifiers (AIDs) and transport address, and provides a *white page* service. Each agent must register with an AMS in order to get a valid AID. The DF provides *yellow pages* services to other agents. Agents register their services with the DF or query the DF to find out what services are offered by other agents.

Deployers of agent platforms specify policies for accessing the agent platform. Specifically, a deployer describes policies for restricting access to four main actions: registration with the AMS, querying the AMS, registration with the DF, and querying the DF.

An agent specifies policies that a requester must satisfy in order to use its service. An agent can either use an internal policy engine module or trust the platform with which it is registered to enforce its policy.

Security is enforced at two levels: platform or agent. In *platform security*, the AMS and DF have additional security features. The AMS decides whether or not to allow an agent to register, search or use its other functions. Similarly, the DF also decides whether to allow an agent to register, modify or search for agents based on certain access control information. An agent, while registering, sends some access control information to the AMS specifying its security category; *private*, *secure* or *open*. A *private* agent's Agent Identifier (AID) is not displayed to any other agent by the AMS, a *secure* agent has to send some access control information so that the AMS can filter requests to the agent and *open* agent is visible to all agents. Similarly, while registering its services with a DF, the agent chooses a category for each service. For example, an agent A can register as an *open* agent with the AMS and register two services with the DF, an open GPS service and a secure navigator service. Agent A also specifies that only agents with certain credentials can access the navigator service. Using these policies, the AMS and the DF decide whether or not to provide a requesting entity information about the matched agent or service, depending on whether the entity meets the matched agent's/service's requirements. In *agent security*, the agent uses its own policy to decide how to further authenticate agents and how to authorize their service requests. The handling of queries by platforms to secure agents is illustrated in Figure 1.

If an agent cannot use the AMS/DF for making authorization decisions on its behalf, then the agent should contain a security/trust module to interpret its own policies. This makes the presence of the reasoning engine in an agent *optional*, allowing agents to run on smaller, lightweight devices. The AMS/DF has a list of conditions that an agent must satisfy in order to contact a particular agent or use a particular service. However, the AMS and DF do not have the agent's policy or have access to its knowledge base. It is upto the agent to make sure that these conditions are accurate and conform to its policy. If the AMS or DF is unable to understand the associated conditions, then based on the policy

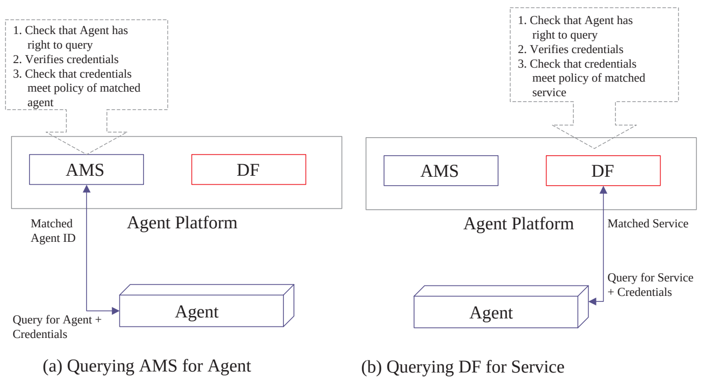
1. Check that Agent has right to query
2. Verifies credentials
3. Check that credentials meet policy of matched agent
AMS
DF
Agent Platform
Matched Agent ID
Agent
Query for Agent + Credentials
(a) Querying AMS for Agent
1. Check that Agent has right to query
2. Verifies credentials
3. Check that credentials meet policy of matched service
AMS
DF
Agent Platform
Matched Service
Query for Service + Credentials
Agent
(b) Querying DF for Service

Fig. 1. An entity that queries a platform must satisfy the matched agent/service policy in order to get access to the agent/service.

of the platform, the AMS and DF can decide to reject all requests for the agent or service or accept all requests and forward them to the agent for interpretation.

6.3 Web Services

Web services generally register themselves with a matchmaker [34] that interprets their functional description and provides brokering services for them. Similar to our agent security framework, web services either use an internal policy engine module to enforce their policy or register their policies along with their functional description with a matchmaker. If it is capable of providing security for itself, a web service filters all requests coming to it by verifying that they match its policy. If a web service does not have a policy engine, the matchmaker is responsible for filtering requests for the web service and only forwards those requests that meet the policy of the web service.

During registration, the web service sends the matchmaker its security policy along with its functional description. The matchmaker may have a policy on which credentials are required by a service to register with it. If the web service meets the matchmaker's requirements, the registration is permitted and the matchmaker stores the service's policy. On receiving a query that matches the functional requirements of the web service, the matchmaker checks if the requester's credentials/properties match the policy of the matched service. If the credentials do not match, the web service is not considered as an answer for the request. The web service can also use an internal policy engine to further filter requests received from the matchmaker. If the web service does not want to use the matchmaker for security, the web service does not register its security policy. In this case, the matchmaker allows all requests that match web service's functionality to go through, and the service decides whether or not to accept it. Figure 2 illustrates the integration of security in a web service environment.

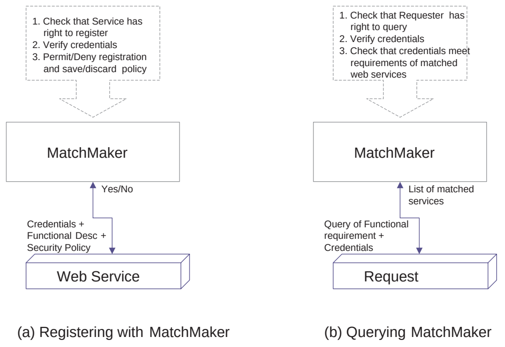
1. Check that Service has right to register
2. Verify credentials
3. Permit/Deny registration and save/discard policy
MatchMaker
Yes/No
Credentials + Functional Desc + Security Policy
Web Service
(a) Registering with MatchMaker
1. Check that Requester has right to query
2. Verify credentials
3. Check that credentials meet requirements of matched web services
MatchMaker
List of matched services
Query of Functional requirement + Credentials
Request
(b) Querying MatchMaker

Fig. 2. The web service registers its functional description and security policy with a matchmaker. When the matchmaker receives a request that matches the web service, it verifies that the request meets the policy of the web service.

This framework is based on our earlier work, Vigil, a security framework, which provides access control to services in pervasive environments [22]. In Vigil, services register their functional description and security policy with the Service Manager of the environment. The Service Manager provides functional and security matchmaking between services and mobile users. Only users that meet a service's security requirements are allowed to view its interface or access it.

7 Current Work and Summary

This paper investigates security issues related to the Semantic Web. We believe that to secure the Semantic Web two main components are required: a semantic policy language for defining security requirements and a distributed policy management approach. In distributed policy management, every entity is able to specify its own policy for security and privacy. It is important for web entities to be able to express their security policies in a clear and concise manner so that its meaning is unambiguous. Towards this end we use a policy language based on a semantic language to markup security information for web entities. Given the inherently distributed nature of the semantic web, the framework includes speech acts like delegation, revocation, request and cancel that dynamically modify existing policies. The policy engine includes strong delegation management making it useful for dynamic systems, consisting of transient resources and users, and distributed systems, in which creating comprehensive policies may be time consuming.

We have developed the specifications of the policy language and a policy engine that reasons over policies in logic and RDF-S, accepts domain dependent information and speech acts, and answers security related queries. We have designed and developed two similar policy based security frameworks, (i) for pervasive environments [22] and (ii) for a supply chain management system [24], which was part of the IBM EECOMS project [35]. We have designed the security framework models for the different web entities including the interaction protocols, and are in the process of developing a security prototype for an agent framework [36].

In this paper, we described the specifications of a semantic policy language and the design of security models for web entities based on this language. We demonstrated through examples the usefulness of our policy language and of our framework based on distributed policy management. We believe that this methodology supports a stronger and more flexible base for securing web entities compared to conventional security mechanisms.

References

1. Cost, R., Finin, T., Joshi, A., Peng, Y., Nicholas, C., Soboroff, I., Chen, H., Kagal, L., Perich, F., Zou, Y., Tolia, S.: ITTALKS: A Case Study in DAML and the Semantic Web. IEEE Intelligent Systems Special Issue 2002 (2002)
2. Horrocks, I., van Harmelen, F., Patel-Schneider, P., Berners-Lee, T., Brickley, D., Connolly, D., Dean, M., Decker, S., Fensel, D., Fikes, R., Hayes, P., Heflin, J., Hendler, J., Lassila, O., McGuinness, D., Stein, L.A.: DAML+OIL Language Specifications. http://www.daml.org (2002)
3. Kopena, J.: DAMLJessKB. (http://plan.mcs.drexel.edu/projects/legorobots/design /software/DAMLJessKB/)
4. Dean, M., Barber, K.: DAML Crawler. (http://www.daml.org/crawler/)
5. Kalyanpur, A., Hendler, J.: RDF Web Scraper Version 1.1. http://www.ece.umd.edu/ adityak/running.html (2002)
6. Kogut, P., Holmes, W.: AeroDAML: Applying Information Extraction to Generate DAML Annotations from Web Pages. (In: First International Conference on Knowledge Capture (K-CAP 2001) Workshop on Knowledge Markup and Semantic Annotation, Victoria, 2001)
7. Eastlake, D., Reagle, J., Solo, D.: XML-Signature Syntax and Processing. (RFC 3275, March 2002)
8. Eastlake, D., Reagle, J.: XML Encryption Syntax and Processing. (W3C Candidate Recommendation, August 2002)
9. Housley, R., Polk, W., Ford, W., Solo, D.: Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. (RFC 3280, April 2002)
10. W3C: Extensible Markup Language. (W3C Recommendation, http://www.w3c.org/XML/)
11. Godik, S., Moses, T.: Oasis extensible access control markup language (xacml). (OASIS Committee Secification cs-xacml-specification-1.0, November 2002)
12. Cranor, L., Langheinrich, M., Marchiori, M., Presler-Marshall, M., Reagle, J.: Platform for privacy preferences (p3p) (2002)
13. Ferraiolo, D., Kuhn, R.: Role-based access controls. In: 15th NIST-NCSC National Computer Security Conference. (1992) 554–563
14. Guiri, L.: A new model for role-based access control (1995)
15. Sandhu, R.S.: Role-based access control. In Zerkowitz, M., ed.: Advances in Computers. Volume 48. Academic Press (1998)
16. Yialelis, N., Lupu, E., Sloman, M.: Role-based security for distributed object systems (1996)

17. Barka, E., Sandhu, R.: Framework for role-based delegation models. In: Annual Computer Security Applications Conference. (2000)
18. Herzberg, A., Mass, Y., J.Mihaeli, D.Naor, Ravid, Y.: Access control meets public key infrastructure : Or assigning roles to strangers. In: 2000 IEEE Symposium on Security and Privacy, Oakland, May 2000. (2000)
19. Hildmann, T., Barholdt, J.: 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)
20. Blaze, M., Feigenbaum, J., Ioannidis, J., Keromytis, A.: The keynote trust management system version (1999)
21. Keromytis, A., Ioannidis, S., Greenwald, M., Smith, J.: The strongman architecture. In: Third DARPA Information Survivability Conference and Exposition (DISCEX III), Washington, D.C. April 22-24. (2003)
22. Undercoffer, J., Perich, F., Cedilnik, A., Kagal, L., Joshi, A., Finin, T.: 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)
23. Kagal, L., Finin, T., Joshi, A.: Trust based security for pervasive computing viroments. In: IEEE Communications, December 2001. (2001)
24. Kagal, L., Finin, T., Peng, Y.: A Framework for Distributed Trust Management. In: Proceedings of IJCAI-01 Workshop on Autonomy, Delegation and Control. (2001)
25. W3C: Resource Description Framework. W3C Recommendation, http://www.w3.org/TR/rdf-schema/ (2002)
26. Swedish Institute, S.I.o.C.S.: SICStus Prolog. http://www.sics.se/ sicstus/ (2001)
27. Kagal, L., Finin, T., Joshi, A.: A policy language for pervasive systems. In: Fourth IEEE International Workshop on Policies for Distributed Systems and Networks. (2003)
28. Moffett, J., Sloman, M.: Policy conflict analysis in distributed systems management. Journal of Organizational Computing (1993)
29. Berners-Lee, T.: Notation 3. http://www.w3.org/DesignIssues/Notation3 (2001)
30. Lupu, E.C., Sloman, M.: Conflicts in policy-based distributed systems management. IEEE Transactions on Software Engineering (1999)
31. Denker, G.: Access control and data integrity for daml+oil and daml-s. White paper (2002)
32. Chodhari, A., Kagal, L., Joshi, A., Finin, T., Yesha, Y.: Patientservice : A policy based information service for ehealthcare. In: Fifth International Workshop on Enterprise Networking and Computing in Healthcare Industry, Santa Monica, June. (2003)
33. FIPA: FIPA Agent Management Specification. In: FIPA website (http://www.fipa.org/specs/fipa00023/). (2001)
34. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.: Semantic matching of web services capabilities. In: ISWC2002. (2002)
35. Ingersoll Rand (Woodcliff Lake, NJ) and QAD (Carpenteria, CA) and Berclain Group (Schaumburg, IL) and IBM Corporation (Somers, NY): CIIMPLEX Consortium, Consortium for Integrated Intelligent Manufacturing PLanning and EXecution. http://www.ciimplex.org (2000)
36. Kagal, L., Finin, T., Joshi, A.: Developing secure agent systems using delegation based trust management. In: Security of Mobile MultiAgent Systems held at Autonomous Agents and MultiAgent Systems. (2002)