Interleaving Semantic Web Reasoning and Service Discovery to Enforce Context-Sensitive Security and Privacy Policies

Norman Sadeh and Jinghai Rao

School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue,
Pittsburgh, PA, 15213 – USA
{sadeh, jinghai}@cs.cmu.edu

Abstract

In many domains, users and organizations need to protect their information and services subject to policies that reflect dynamic, context-sensitive considerations. More generally, enforcing rich policies in open environments will increasingly require the ability to dynamically identify external sources of information necessary to enforce different policy elements (e.g. finding an appropriate source of location information to enforce a location-sensitive access control policy). In this paper, we introduce a semantic web framework for dynamically interleaving policy reasoning and external service discovery and access. Within this framework, external sources of information are wrapped as web services with rich semantic profiles allowing for the dynamic discovery and comparison of relevant sources of information. Each entity (e.g. user, sensor, application, or organization) relies on one or more Policy Enforcing Agents responsible for enforcing relevant privacy and security policies in response to incoming requests. These agents implement meta-control strategies to dynamically interleave semantic web reasoning and service discovery and access. This research has been conducted in the context of *my*Campus, a pervasive computing environment aimed at enhancing everyday campus life at Carnegie Mellon University.

1. Introduction

The increasing reliance of individuals and organizations on the Web to help mediate a variety of activities is giving rise to a demand for richer security and privacy policies and more flexible mechanisms to enforce these policies. Enterprises want to selectively expose core business functionality and sensitive business information to various partners based on the evolving nature of their relationships (e.g. disclosing rough product specifications to prospective suppliers versus disclosing more detailed requirements to actual suppliers, or giving selective visibility into the company's inventory positions or demand forecasts to preferred supply chain partners). Employees in a company may be willing or required to share information about their whereabouts or about their activities with some of their team members or their boss but only under some conditions (e.g. during regular business hours or while on company premises). Coalition forces may need to selectively share sensitive intelligence information but only to the extent it is relevant to a specific joint mission. Each of these examples illustrates the need for what we generically refer to as context-sensitive security and privacy policies, namely policies whose conditions are not tied to static considerations but rather conditions whose satisfaction, given the very same actors (or principals), will likely fluctuate over time. Enforcing such policies in open environments is particularly challenging for several reasons:

- Sources of information available to enforce these policies may vary from one principal to another (e.g. different users may have different sources of location tracking information made available through different cell phone operators)
- Available sources of information for the same principal may vary over time (e.g. when a user is on company premises her location may be obtained from the wireless LAN location tracking functionality operated by her company as well as through her cell phone operator, but when she is not on company premises the cell phone operator is the only option – subject to relevant privacy policies she may have specified)
- Available sources of information may not be known ahead of time (e.g. new location tracking functionality may be installed or the user might roam into a new area)

Accordingly, enforcing context-sensitive policies in these open domains requires the ability to opportunistically interleave policy reasoning with the dynamic identification, selection and access of relevant sources of contextual information. This requirement exceeds the capability of decentralized management infrastructures proposed so far and calls for privacy and security enforcing mechanisms capable of operating according to significantly less scripted scenarios than is the case today (e.g. [BSF02,HSSK04,LGC+05]). It also calls for much richer

service profiles than those found in early web service standards.

In this paper, we introduce a semantic web framework for dynamically interleaving policy reasoning and external service identification, selection and access. Within this framework, external sources of information are wrapped as web services with rich semantic profiles allowing for the dynamic discovery and comparison of relevant sources of information. Each entity (e.g. user, sensor, application, or organization) relies on one or more Policy Enforcing Agents responsible for enforcing relevant privacy and security policies in response to incoming requests. These agents implement meta-control strategies to opportunistically interleave semantic web reasoning and service discovery and access. In this paper, we focus on a particular type of Policy Enforcing Agent we refer to as Information Disclosure Agent. These agents are responsible for enforcing two types of policies: access control policies and obfuscation policies. The latter are policies that manipulate the accuracy or inaccuracy with which information is released (e.g. disclosing whether someone is busy or not rather than disclosing what they are actually doing). The research reported herein has been conducted in the context of MyCampus, a pervasive computing environment aimed at enhancing everyday campus life at Carnegie Mellon University [SCV+03,GS03,GS04a].

The work presented in this paper builds on concepts of decentralized trust management developed over the past decade [BFL96]. Most recently, a number of researchers have started to explore opportunities for leveraging the openness and expressive power associated with Semantic Web and agent frameworks in support of decentralized trust management (e.g. [UPC+03, KFJ03, KPS04, HKL+04, APM04, UBJ04, DKF+05] to name just a few). Our own work in this area has involved the development of Semantic e-Wallets that enforce context-sensitive privacy and security policies in response to requests from context-aware applications implemented as intelligent agents [GS03, GS04a]. In this paper, we introduce a significantly more decentralized framework, where policies can be distributed among any number of agents and web services. Within this framework, we present a meta-control architecture for interleaving semantic web reasoning and web service discovery in enforcing context-sensitive privacy and security policies.

The remainder of this paper is organized as follows. Section 2 introduces an overall architecture for distributing and enforcing privacy and security policies, using a pervasive computing context to illustrate how these policies can be deployed in practice. It follows with an overview of our Information Disclosure Agent, detailing its different modules and how their operations are opportunistically orchestrated in response to incoming requests. A motivating example based on the pervasive computing environment introduced earlier is presented in Section 3. Section 4 details our query status model, which serves as a basis to our meta-control strategies. Section 5 describes our service discovery model. Some implementation issues are discussed in Section 6. Concluding remarks are provided in Section 7.

2. Overall Approach and Architecture

Pervasive Computing as an Application Context

To help put things in perspective, we consider a pervasive computing environment, where each user interacts with the infrastructure either directly (e.g. walking into a room, entering the subway system) or indirectly via agents to which they delegate tasks (e.g. a general-purpose user-agent such as a micro-browser on a PDA or cell phone, policy evaluation and notification agents, or task-specific agents such as a context-aware message filtering agent or a meeting scheduler agent). The infrastructure provides a set of resources generally tied to different geographical areas, such as printers, surveillance cameras, campus-based location tracking functionality, and so on (see Figure 1). These resources are all modeled as services that can be automatically discovered based on rich ontology-based service profiles advertised in service directories and accessed via open APIs. In general, services can offer functionality and/or serve as sources of contextual information. A camera service, a calendar system, or a location tracking service are examples that can offer both. Services can also build on one another, with simple services providing building blocks for the definition of more complex ones. An example is the “printer service” in Figure 1, which itself relies on the “find nearest printer service,” which in turn relies on a “people locator service” to find the location of the user. The “people locator service” in turn might be able to dynamically select from a number of possible services available to locate people such as a badge system or a combination of a system of cameras along with a video analysis service. Each service and agent has an owner, whether an individual or an organization, who is responsible for setting policies for the service or agent.

Services that collect information about users may broadcast disclosure messages that inform target users (or more specifically their agents) about the operation of the service (e.g. users who enter a smart room or the subway system). Some disclosures are one-way announcements: they simply inform the user that information is collected about them and possibly how that information is used.

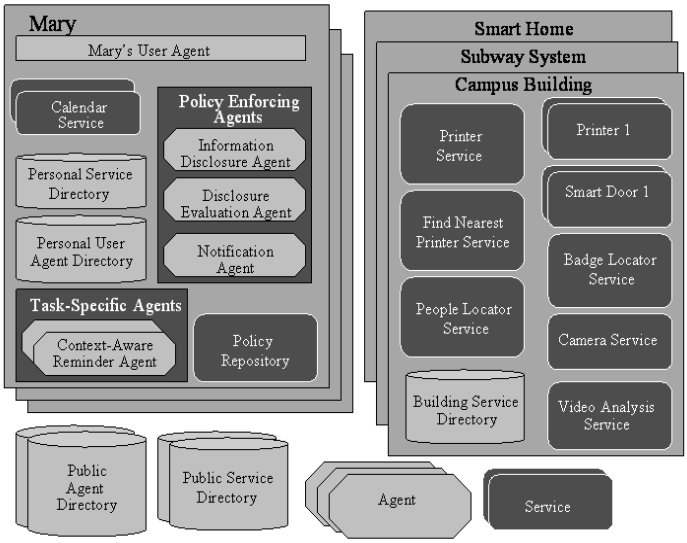

Figure 1. Pervasive Computing as an Application Domain

Other disclosure messages may give the user some options. For example, a location-tracking service may give the user the choice of opting out. Alternatively, the user may be able to allow tracking, while limiting the use of his or her location information (e.g. only for emergency use) or she may require that all requests for her location be cleared with her own Information Disclosure Agent (enforcing her regular privacy and security policies). A Policy Disclosure Evaluation Agent may respond to disclosures automatically, based on the user's policies (e.g. opting out). The same agent may also be able to occasionally notify its user of policies that might lead her to modify her behavior, as well as prompt its user to manually select among possible options when needed.

Each entity (or principal) in the system (whether an individual, a service, an agent or an organization) has a set of credentials and a set of policies. These policies can include:

- Access control policies that limit access only to entities that can be proved to satisfy certain conditions.
- Obfuscation policies that associate different levels of accuracy or inaccuracy to different sets of credentials.
- Information collection policies (a la P3P [CLM+02], that specify what type of information is collected by a service, for what purpose, how that information will be stored, etc.
- Notification Preference Policies specifying under which conditions a user may want to be alerted about the presence of sensors or other information collection applications.

Collectively, these policies enable users and organizations to manage their privacy practices, specifying what information they are willing to disclose (access control) and at what level of granularity (obfuscation) and notifying users or their agents about the information they collect and what happen to that information. Policy enforcement is delegated to different sets of agents (these agents may occasionally request input or feedback from their users, as already illustrated earlier). For the sake of clarity, in the remainder of this paper, we focus more specifically on one such type of agent, namely an *Information Disclosure Agent* responsible for enforcing both access control and obfuscation policies. The architecture presented for this agent can however be adapted to implement a number of other context-sensitive Policy Enforcing Agents such as the ones illustrated in Figure 1.

Information Disclosure Agent: An Example of a Policy Enforcing Agent

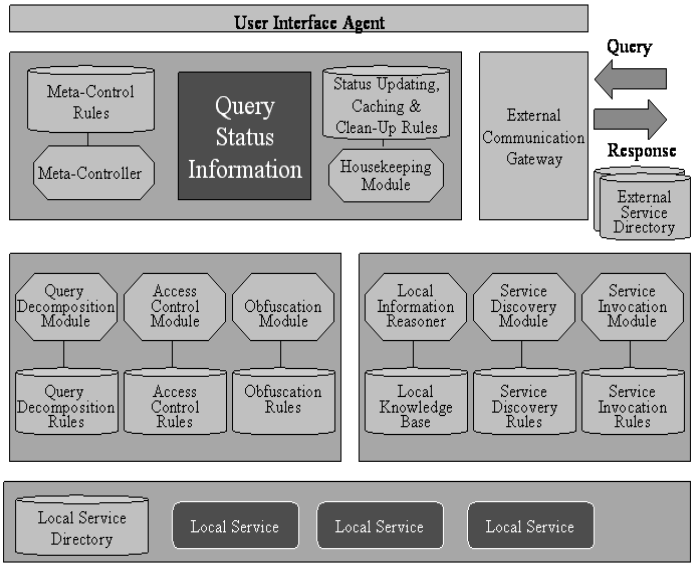

Figure 2. Information Disclosure Agent: Logical Architecture

An Information Disclosure Agent (IDA) processes incoming requests (e.g. a query about the location of the agent's owner or a request to access a service under the owner's control) subject to a set of access control and obfuscation polices captured in the form of rules. As it processes incoming queries, the agent records status information that helps it monitor its own progress in enforcing its policies and in obtaining the necessary information. Based on this updated query status information, a meta-control module ("meta-controller") dynamically orchestrates the operations of modules it has at its disposal to process queries (Figure 2). As these modules report on the status of activities they have been tasked to perform, this information is processed by a Housekeeping module responsible for updating query status information (e.g. changing the status of a query from being processed to having been processed). Simply put, the

agent continuously cycles through the following three basic steps:

1. The meta-controller analyzes the latest query status information and invokes one or more modules to perform particular tasks. As it invokes these modules the meta-controller also updates relevant query status information (e.g. update the status of a query from "not yet processed" to "being processed"). All query status information includes timestamps.
2. Modules complete their tasks (whether successfully or not) and report back to the Housekeeping module – occasionally modules may also report on their ongoing progress in handling a task
3. The Housekeeping module updates detailed status information based on information received from modules and performs additional housekeeping activities (e.g. caching the results of recent requests to mitigate the effects of possible denial of service attacks, cleaning up status information that has become irrelevant, etc.)

For obvious efficiency reasons, while an IDA consists of a number of logical modules, each operating according to a particular set of rules, it is actually implemented as a single reasoning engine. In our current work we use JESS [Fri03], a high-performance Java-based rule engine that supports both forward and backward chaining – the latter by reifying "needs for facts" as facts themselves, which in turn trigger forward-chaining rules. The following provides a brief description of each of the modules orchestrated by the IDA's meta-controller – note that other types of Policy Enforcing Agents typically entail different sets of modules:

- *Query Decomposition Module:* This module takes as input a particular query an breaks it down into elementary needs for information, which can each be thought of as subgoals or sub-queries. We refer to these as *Query Elements*.
- *Access Control Module* is responsible for determining whether a particular query or sub-query is consistent with relevant access control policies – modeled as access control rules. While some policies can be checked just based on facts contained in the agent's local knowledge base, many policies require obtaining information from a combination of both local and external services. When this is the case, rather than immediately deciding whether or not to grant access to a query, the Access Control Module requests additional facts – *also Query Elements*. These requests are added to the agent's Query Status Information Knowledge Base along with information about their parent Query or Query Element – namely the Query or Query Element for which they are needed.
- *Obfuscation Module* sanitizes information requested in a query according to relevant obfuscation policies – also modeled as rules. As it evaluates relevant obfuscation policies, this module too can post request for additional information (Query Elements) to the Query Status Information Knowledge Base (via the Housekeeping Module).
- *Local Information Reasoner*: This reasoner corresponds to "static" domain knowledge (facts and rules) known locally to the IDA or at least knowledge that does not change too frequently (e.g. the name and email address of the agent's owner, possibly a list of friends and family members, etc.)
- *Service Discovery Module*: This module helps the IDA identify promising sources of information to complement its local knowledge. This includes both *local services* and *external services*. Local services can be identified through a local service directory (e.g. a directory of services under the direct control of the agent's owner such as a calendar service running on his desktop or on his smart phone). External services can be identified through external service directories (whether public or not). Communication with external service directories takes place via the agent's *External Communication Gateway*. Rather than relying solely on searching service directories, the service discovery module also allows for the specification of what we refer to as *service identification rules*. These rules directly map information needs on prespecified services (whether local or external). An example of such rule might be: "when looking for current activity, try first my calendar service". When available, such rules can yield significant performance improvements, while allowing the module to revert to more general service directory searches when they fail. We assume that all service directories rely on OWL-S to advertise service profiles (See Section 5).
- *Service Invocation Module:* This module allows the agent to invoke relevant services, whether local or external. It is important to note that, in our architecture, each service can have its own Information Disclosure Agent (IDA). As requests are sent to services, their IDAs may in turn respond with requests for additional information to enforce their own policies.
- *User Interface Agent:* The meta-controller treats its user as just another module who is modeled both as a potential source of domain knowledge (e.g. to acquire relevant contextual information) as well as a potential source of meta-control knowledge (e.g. if a particular query takes too long to process, the user may be requested whether it is worth expending additional computational resources processing that query or not).

Modules support one or more services that can each be invoked by the meta-controller along with relevant parameter values. For instance, the meta-controller may invoke the query decomposition module and request it to decompose a particular query; it may invoke the access

control module and task it to proceed in evaluating access control policies relevant to a particular query; etc. In addition, meta-control strategies do not have to be sequential. For instance, it may be advantageous to implement *meta-control strategies* that enable the IDA to concurrently request the same or different facts from several services..

3. Sample Scenario

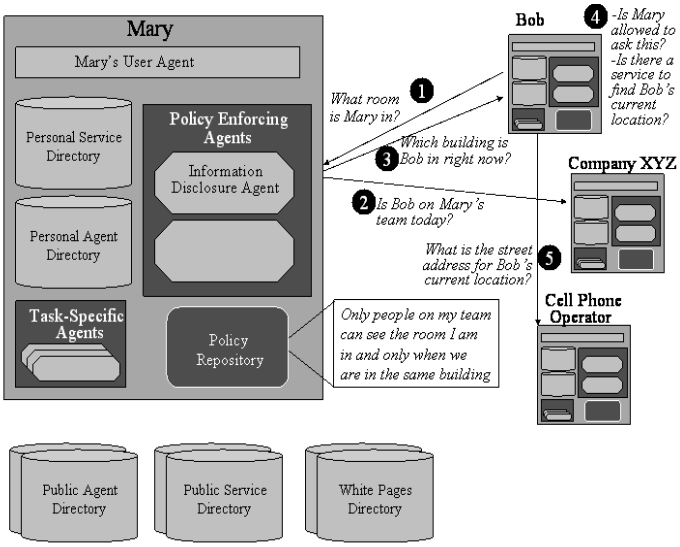

Figure 3. Illustration of first few steps involved in processing a request from Bob to find out about the room Mary is in.

The following scenario will help illustrate how IDAs operate. Consider Mary and Bob, two colleagues who work for Company XYZ. Mary and Bob are both field technicians who constantly visit other companies. Mary's team changes from one day to the next depending on the nature of her assignment. Mary relies on an Information Disclosure Agent to enforce her access control policies. In particular, she has specified that she is only willing to disclose the room that she is in to members of her team and only when they are in the same building. Suppose that today Bob and Mary are on the same team and that Bob is querying Mary's IDA to find out about her location. For the purpose of this scenario, we assume that Mary and Bob are visiting Company ABC and are both in the same building at the time the query is issued. Both Bob and Mary have cell operators who can provide their location at the level of the building they are in – but not at a finer level. Upon entering Company ABC, Mary also registered with the company's location tracking service, which operates over the wireless LAN and is compatible with her WiFi-enabled smart phone. As she registered with the service, one of her Policy Enforcing Agents (her Policy Disclosure Evaluation Agent) negotiated that all requests about her location be redirected to her IDA. For the purpose of this scenario, we also assume that Mary's IDA does not yet know whether Bob is on her team. It therefore needs to identify a service that can help it determine whether this is the case. A service discovery step helps identify a service operated by Company XYZ (Bob and Mary's employer) that contains up-to-date information about teams of field technicians. This step requires a directory with rich semantic service profiles, describing what each service does (e.g. type of information it can provide, level of accuracy or recency, etc.). To be interpretable by agents such as Mary's IDAs, these profiles also need to refer to concepts specified in shared ontologies (e.g. concepts such as projects, teams, days of the week, etc.). Once Mary's IDA has determined that Bob is on her team today, it proceeds to determine whether they are in the same building by asking Bob's IDA about the building he is in. Here Bob's IDA goes through a service discovery step of its own and determines that a location tracking service offered by his cell phone operator is adequate. Completion of the scenario involves a few additional steps of the same type. Note that in this scenario we have assumed that Mary's IDA trusts the location information returned by Bob's IDA. It is easy to imagine scenarios where her IDA would be better off looking for a completely independent source of information. It is also easy to see that these types of scenarios can also lead to deadlocks. In later sections, we briefly discuss elements of our architecture that partially helps mitigate these problems (e.g. query status update information that keeps track of the origin of requests for information – see the section below).

4. Query Status Model

The IDA's *Meta Controller* relies on meta-control rules to analyze query status information and determine which module(s) to activate next. Meta-control rules are currently modeled in CLIPS. In other words, each meta-control rule is an if-then clause, with a LHS (left hand side) specifying its premises and a RHS (right hand side) its conclusions. More specifically, LHS elements of meta-control rules refer to query status information, while RHS ones contain facts that result in module activations. While both LHS and RHS are expressed in CLIPS they refer to queries received by the IDA and to *query elements* generated while processing these queries. A query element is a need for elementary information required to fully process a query (e.g. finding someone's location or calendar activity to help answer a more complex query). Queries themselves are expressed in an extension of OWL (see [GS04a]). Query status information in the LHS relies on a taxonomy of predicates that helps the agent keep track of queries and query elements - e.g., whether a query has been or is being processed, what individual query elements it has given rise to, whether these elements have been cleared by relevant access control policies and sanitized

according to relevant obfuscation control policies. Query status information helps keep track of how far along the IDA is in obtaining the information required by each query element, whether the agent's local knowledge base has been consulted, whether local or external services have been identified and consulted, etc. It also enables the agent to keep track of dependencies between queries and query elements. This information can help identify potential deadlocks. All query status information is time stamped, enabling the meta-controller to also implement rules that take into account how much time has already been spent trying to process a query, clearing access control policies or waiting for an external service to respond. A sample of query status information predicates is provided in Table 1. This list is just illustrative and will be used to revisit the scenario introduced earlier. Clearly, different taxonomies of predicates can lead to more or less sophisticated meta-control strategies. For the sake of clarity, status predicates in Table 1 are organized in six categories: 1) communication; 2) query; 3) query elements; 4) access control; 5) obfuscation and 6) information collection. Status information is represented in CLIPS with status predicates and a number of slots detailing particular pieces of status information. Typical slots include:

- **A query ID** or **query element ID** to which the predicate refers
- **A parent query ID** or **parent query element ID** to help keep track of dependencies (e.g. a query element may be needed to help check whether another query element is consistent with a context-sensitive access control policy). These dependencies, if passed between IDA agents, can also help detect deadlocks (e.g. two IDA agents each waiting for information from the other to enforce their policies)
- **A time stamp** that describes when the status information was generated or updated. This information is critical when it comes to determining how much time has elapsed since a particular module or external service was invoked. It can help the agent look for alternative external services or decide when to prompt the user (e.g. to decide whether to wait any longer).

	Sample Status Predicates	**Description**
1)	Query-Received	Query received. A related queries slot helps determine the query's context and identify potential deadlocks.
	Sending-Response	Response to a query is being sent
	Response-Sent	Response has been successfully sent
	Response-Failed	Response failed (e.g. message bounced back)
2)	Processing Query	Query is being processed
	Query Decomposed	Query has been decomposed (into primitive query elements)
	All-Elements-Available	All query elements are available (i.e. the information they require is available)
	All-Elements-Cleared	All query elements have been cleared by relevant access control policies
	Clearance-Failed	Failed to clear one or more access control policies
	All-Elements-Sanitized	All query elements have been sanitized according to relevant obfuscation policies
	Sanitization-Failed	Failed to pass one or more obfuscation policies
3)	Element-Needed	A query element is needed. Query elements may result from the decomposition of a query or may be needed to enforce policies. The query element's origin helps distinguish between these different cases
	Processing-Element	A need for a query element is being processed
	Element-Available	Query element is available
	Element-Cleared	Query element has been cleared by relevant access control policies
	Clearance-Failed	Failed to pass one or more access control policies
	Element-Sanitized	Query element has been sanitized according to relevant obfuscation policies
	Sanitization-Failed	Failed to pass one or more obfuscation policies
4)	Clearance-Needed	A query or query element needs to be cleared by relevant access control rules
5)	Sanitization-Needed	Query or query element has to be sanitized subject to relevant obfuscation policies
6)	Check-Condition	Check whether a condition is satisfied. Special type of query element.
	Element-not-locally-available	The value of a query element can not be obtained from the local knowledge base
	Element-need-service	A query element requires the identification of a relevant service
	No-service-for-Element	No service could be identified to help answer a query element. This predicate can be refined to differentiate between different types of services (e.g. local versus external)
	Service-identified	One or more relevant services have been identified to help answer a query element
	Waiting-for-service-response	A query element is waiting for a response to a query sent to a service (e.g. query sent to a location tracking service to help answer a query element corresponding to a user's location)
	Failed-service-response	A service failed to provide a response. Again this predicate could be refined to distinguish between different types of failure (e.g. service down, access denied, etc.)
	service-response-available	A response has been returned by the service. This will typically result in the creation of an "Element-Available" status update.

Table 1. Sample list of status information predicates.

Query status information updates are asserted as new facts (with old information being cleaned up by the IDA's housekeeping module – Figure 2). As query updates come in, they trigger one or more meta-control rules, which in turn result in additional query status information updates and the eventual activation of one or more of the IDA's modules. An example of a simple meta-control rule to activate the service discovery module if information about the room that Mary could not be obtained locally (from the local information reasoner) can be expressed as follows:

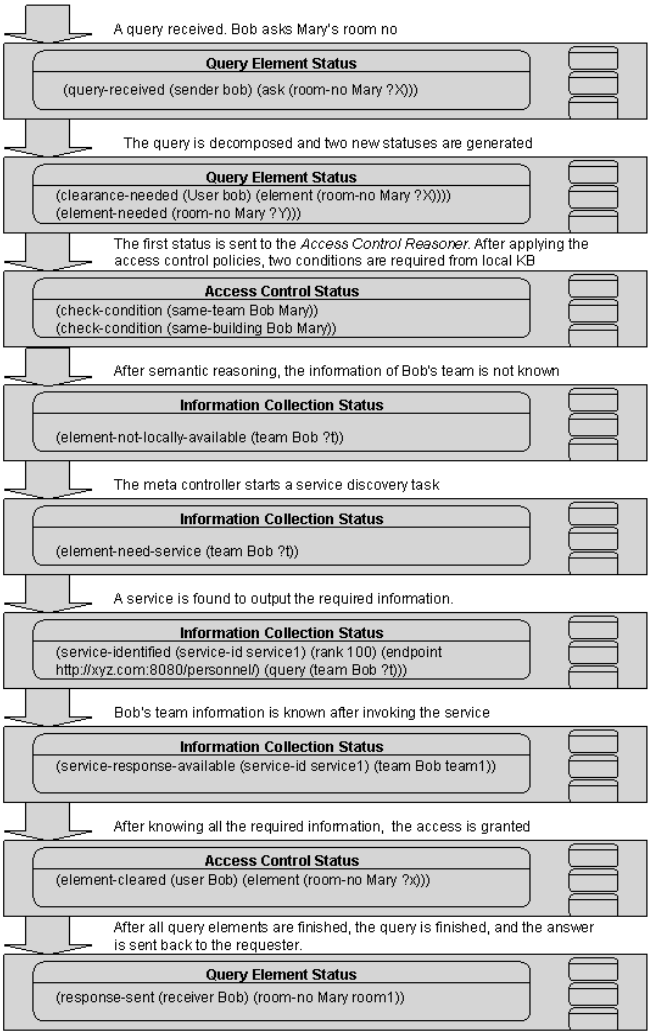

Figure 4. An example of status changes

```
(element-needed (parent-id ?x) (elem-id ?y) (room
Mary ?z))
(element-not-locally-available (elem-id ?y) (room
Mary ?z1))
=>
(assert (module service-discovery) (element-need-
service (elem-id ?y) (output (room Mary ?z))))
```

In practice, meta-control rules are typically more general than this (i.e. they don't just refer to the room Mary is in).

Example

The following illustrates the processing of a query by an IDA, using the scenario introduced in Figure 3. Figure 4 depicts some of the main steps involved in processing a request from Bob about the room Mary is in, highlighting some of the main query status information updates. Specifically, Bob's query about the room Mary is in is processed by the IDA's *Communication Gateway*, resulting in a query information status update indicating that a new query has been received:

```
(query-received (queryid 1) (sender Bob) (ask
(room-no Mary ?X)))
```

The meta-controller proceeds by invoking the *Query Decomposition Module*, resulting in the creation of two query elements – for the sake of simplicity we omit Mary's obfuscation policy: one to establish whether this request is compatible with Mary's access control polcies and the other to obtain the room she is in:

```
(clearance-needed (parent-id 1) (elem-id 1.1)
(User Bob) (element (room-no Mary ?x)))
(element-needed (parent-id 1) (elem-id 1.2)
(room-no Mary ?X))
```

The meta-controller decides to first focus on the "clearance-needed" query element and invokes the *Access Control Module*. This module determines that two conditions need to be checked and accordingly creates two new query elements ("check-conditions"):

```
(check-condition (parent-id 1.1) (elem-id
1.1.1) (same-team Bob Mary) )
(check-condition (parent-id 1.1) (elem-id
1.1.1) (same-building Mary Bob))
```

The first condition requires checking whether Bob and Mary are on the same team, while the second one is to determine whether Bob is in the same building as Mary. Each condition requires a series of information collection steps that are orchestrated by the meta-control rules in Mary's IDA. In this example, we assume that the IDA's local KB contains a semantic reasoning rule:

```
(team ?p1 ?t)
(team ?p2 ?t)
=>
(same-team ?p1 ?p2)
```

We also assume that the IDA knows Mary's team but not Bob's. According the following query status information update is generated:

```
(element-not-locally-available (parent-id
1.1.1) (elem-id 1.1.1.1) (team Bob ?t))
```

Mary's IDA has a meta-control rule to initiate service discovery when a query element can not be found locally. The rule is of the form:

```
(element-needed (elem-id ?x) ?y)
(element-not-locally-available (elem-id ?x)
?y)
=>
(assert (module discover) (element-need-
service (parent-id ?x) (elem-id ?z) ?y))
```

Thanks to this rule, the *Service Discovery Module* is now activated. A service to find Bob's team is identified (e.g. a

service operated by company XYZ). This results in a Query Status Information update of the type "service-identified". If there are multiple matching services, they may be ranked and the top service is invoked (multiple services could also be invoked concurrently).

```
(service-identified (elem-id ?e) (service-id
?s1) (rank ?r1) (endpoint ?e1) ?x)
(not (service-identified (elem-id ?e)
(service-id ?s2) (rank ?r2) (endpoint ?e2)
?x))
(leq ?r1 ?r2)
=>
(assert (module invocation) (invoke-service
(parent-id ?e) (elem-id ?ee) (service-id ?s1)
(endpoint ?r1) ?x))
```

We assume that the service returns the team that Bob is in. The Housekeeping module updates the necessary Query Status Information, indicating among other things that information about Bob's team has been found ("element-available") and cleaning old status information. This is done using a rule of the type:

```
?n <-(element-needed (elem-id ?e) ?y)
(service-response-available (parent-id ?e)
(elem-id ?ee) (service-id ?s) ?a)
=>
(retract ?n)
(assert (module meta) (element-available
(parent-id ?ee) (elem-id ?eee) ?a))
```

The scenario continues through a number of similar steps.

5. The Service Discovery Model

A central element of our architecture is the ability of IDA agents to dynamically discover sources of information (whether local or external) to help obtain the information needed by Query Elements. Sources of information are modeled as Semantic Web Services and may operate subject to their own access control and obfuscation policies enforced by their own IDA agents. Accordingly service invocation is itself implemented in the form of queries sent to a service's IDA agent.

Service Model

Each service (or source of information) is described by a *ServiceProfile* in OWL-S [W3C04]. *ServiceProfiles* consist of three parts: (1) information about the provider of the service, (2) information about the service's functionality and (3) information about non-functional attributes [SEH02]. Functional attributes include the service's inputs, outputs, preconditions and effects. Non-functional attributes are other properties such as accuracy, quality of service, price, location, etc. An example of a location tracking service operated on the premises of Company ABC is described in Figure 5.

```
<profileHierarchy:InformationService
            rdf:ID="PositioningService ">
  <!-- reference to the service specification -->
  <service:presentedBy
    rdf:resource="&Service;#PositioningService"/>
  <profile:has_process
      rdf:resource="&Process;#PositionProc"/>
  <profile:serviceName>Positioning_Service_in_ABC
  </profile:serviceName>

  <!-- specification of quality rating for
                                    profile -->
    <profile:qualityRating>
      <profile:QualityRating rdf:ID="SERVQUAL">
        <profile:ratingName>
          SERVQUAL
        </profile:ratingName>
      <profile:rating
          rdf:resource="&servqual;#Good"/>
    </profile:QualityRating>

  <profile:hasPrecondition
          rdf:resource="&Process;#LocInABC"/>
  <profile:hasOutput
      rdf:resource="&Process;#RoomNoOutput"/>

</profileHierarchy:InformationService>
```

Fig. 5. An example service profile in OWL-S

Because in our architecture service invocation is done by submitting queries to a service's IDA, our service profiles currently do not include inputs. Instead, services send obtain their input parameters by submitting queries back to the requester. In practice, this process can become somewhat inefficient and we plan to also investigate more sophisticated discovery models that examine required service input requirements in light of the IDA's access control and obfuscation policies.

Service outputs are represented as OWL classes, which play the role of a typing mechanism for concepts and resources. Using OWL also allows for some measure of semantic inference as part of the service discovery process. If an agent requires a service that produces a contextual attribute as output of a specific type, then all services that output the value of that attribute as a subtype are potential matches.

Service preconditions and effects are also used for service matching. For instance., the positioning service in Figure 5 has a precondition specifying that it is only available on company ABC's premises.

6. Implementation

Our policy enforcing agents are based on JESS, a high-performance rule-based engine implemented in Java (see [RS05] for additional details and performance results). Domain knowledge, including service profiles, queries, access control policies and obfuscation policies are expressed in either in OWL or in extensions of OWL

[GS04a]. XSLT transformations are used to translate OWL facts and extensions of OWL (to model rules and queries) into CLIPS . Query status information and meta-control rules are directly expressed in CLIPS. Agent modules are organized as JESS modules. Rules in a JESS module only fire when that module has the focus and only one module can be in focus at a time. Currently all information exchange between agents is done in the clear and without digital signatures. In the future, we plan to use SSL or some equivalent protocol for all information exchange.

7. Conclusion Remarks

In many domains, users and organizations need to protect their information and services subject to policies that reflect dynamic, context-sensitive considerations. More generally, enforcing rich policies in open environments will increasingly require the ability to dynamically identify external sources of information necessary to enforce different policy elements. In this paper, we presented a semantic web framework for dynamically interleaving policy reasoning and external service discovery and access. Within this framework, external sources of information are wrapped as web services with rich semantic profiles allowing for the dynamic discovery and comparison of relevant sources of information. Each entity (e.g. user, sensor, application, or organization) relies on one or more *Policy Enforcing Agents* responsible for enforcing relevant privacy and security policies in response to incoming requests. These agents implement meta-control strategies to dynamically interleave semantic web reasoning and service discovery and access.

The Information Disclosure Agent presented in this paper is just one instantiation of our more general concept of Policy Enforcing Agents. Other policies (e.g. information collection policies, notification preference policies) will typically rely on slightly different sets of modules and different meta-control strategies, yet they could all be implemented using the same type of architecture and many of the same principles presented in this paper. Our Policy Enforcing Agents rely on a taxonomy on query information status predicates to monitor their own progress in processing incoming queries and enforcing relevant security and privacy policies. They use meta-control rules to decide which action to take next (e.g. decomposing queries, seeking local or external information, etc.). This work is conducted in the context of *my*Campus, a context-aware environment aimed at enhancing everyday campus life at Carnegie Mellon University [SCV+03,GS04a]. Experiments with an early implementation of our framework seem promising. At the same time, it is easy to see that the generality of our framework also gives rise to a number of challenging issues. Future work will focus on testing the scalability of our framework, evaluating tradeoffs between the expressiveness of privacy and security policies we allow and associated computational and communication requirements. Other issues of particular interest include studying opportunities for concurrency (e.g. simultaneously accessing multiple web services), dealing with real-time meta-control issues (e.g. deciding when to give up or when to look for additional sources of information/web services), and breaking deadlocks [LNOS04].

Acknowledgements

The work reported herein has been supported in part under DARPA contract F30602-02-2-0035 and in part under ARO research grant DAAD19-02-1- to Carnegie Mellon University's CyLab. Additional support has been provided by IBM, HP, Symbol, Boeing, Amazon, Fujitsu, the EU IST Program (SWAP project), and the ROC's Institute for Information Industry. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon

References

[APM04] R. Ashri, T. Payne, D. Marvin, M. Surridge and S. Taylor, Towards a Semantic Web Security Infrastructure. In *Proceedings of Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series*, Stanford University, Stanford California.

[BSF02] Lujo Bauer, Michael A. Schneider and Edward W. Felten. "A General and Flexible Access Control System for the Web", In *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[BFL96] Matt Blaze, Joan Feigenbaum, an Jack Lacy. "Decentralized Trust Management". Proc. IEEE Conference on Security and Privacy. Oakland, CA. May 1996

[CLM+02] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler Marshall, and J. Reagle. The platform for privacy preferences 1.0 (P3P1.0) Specification. W3C Recommendation, April 16, 2002.

[DKF+05] Li Ding, Pranam Kolari , Tim Finin , Anupam Joshi, Yun Peng and Yelena Yesha. "On Homeland Security and the Semantic Web: A Provenance and Trust Aware Inference Framework", In *Proceedings of the AAAI Spring Symposium on AI Technologies for Homeland Security*, March 2005.

[Fri03] Friedman-Hill, E.: Jess in Action: Java Rule-based Systems, Manning Publications Com-pany, June 2003, ISBN 1930110898, http://herzberg.ca.sandia.gov/jess/

[GPH03] Golbeck, J.; Parsia, B.; and Hendler, J. 2003. Trust networks on the Semantic Web. In *Proceedings of*

7th International Workshop on Cooperative Intelligent Agents, CIA 2003.

[GS03] F. Gandon, and N. Sadeh. A semantic e-wallet to reconcile privacy and context awareness. In *Proceedings of the Second International Semantic Web Conference (ISWC03)*, Florida, October 2003.

[GS04a] F. Gandon, and N. Sadeh. Semantic web technologies to reconcile privacy and context awareness. *Web Semantics Journal*, 1(3), 2004.

[HKL+04] R. Hull, B. Kumar, D. Lieuwen, P. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas. Enabling context-aware and privacy-conscious user data sharing. In *Proceedings of 2004 IEEE International Conference on Mobile Data Management*, Berkeley, California, January 2004.

[HS04] U. Hengartner, and P, Steenkiste. Implementing access control to people location information. In *9th ACM Symposium on Access Control Models and Technologies (SACMAT'04)*, Yorktown Heights, June 2004

[HSSK04] T. van der Horst, T. Sundelin, K. E. Seamons, and C. D. Knutson. Mobile Trust Negotiation: Authentication and Authorization in Dynamic Mobile Networks. *Eighth IFIP Conference on Communications and Multimedia Security*, Lake Windermere, England, September 2004

[KFJ03] L. Kagal, T. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *Collection of IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, June 2003

[KPS04] L. Kagal1, M. Paolucci, N. Srinivasan, G. Denker, T. Finin and K. Sycara, Authorization and Privacy for Semantic Web Services, In *Proceedings of Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series*, Stanford University, California, March 2004.

[LGC+05] L.Bauer, S. Garriss, J. McCune, M.K. Reiter, J. Rouse, and P Rutenbar, “Device-Enabled Authorization in the Grey System”, Submitted to *USENIX Security 2005*. Also available as Technical Report CMU-CS-05-111, Computer Science Department, Carnegie Mellon University, February 2005.

[LNOS04] T. Leithead, W. Nejdl, D. Olmedilla, K. Seamons, M. Winslett, T. Yu, and C. Zhang, How to Exploit Ontologies in Trust Negotiation. *Workshop on Trust, Security, and Reputation on the Semantic Web, part of the Third International Semantic Web Conference*, Hiroshima, Japan, November 2004.

[RS05] J. Rao. and N. Sadeh "*Interleaving Semantic Web Reasoning and Service Discovery to Enforce Context-Sensitive Security and Privacy Policies*", Carnegie Mellon Univ., Sch. of Computer Science Tech. Report CMU-ISRI-TR-05-133, July 2005.

[Rao04] J. Rao. "*Semantic Web Service Composition via Logic-based Program Synthesis*". *PhD Thesis*. Department of Computer and Information Science, Norwegian University of Science and Technology, December 10, 2004.

[RKM04a] J. Rao, P. Küngas and M. Matskin, "Composition of Semantic Web Services using Linear Logic Theorem Proving". *To appear in Information Systems Journal - Special Issue on the Semantic Web and Web Services"*.

[SCV+03] N. M. Sadeh, T.C. Chan, L. Van, O. Kwon, and K. Takizawa. Creating an open agent environment for context-aware m-commerce. *In Agentcities: Challenges in Open Agent Environments*, ed. by Burg, Dale, Finin, Nakashima, Padgham, Sierra, and Willmott, LNAI, Springer Verlag, pp.152-158, 2003.

[SEH02] J. O'Sullivan, D. Edmond, and A. T. Hofstede. What's in a service? Towards accurate description of non-functional service properties. *Distributedand Parallel Databases*, 12:117.133, 2002.

[UPC+03] J. Undercoffer, F. Perich, A .Cedilnik, L. Kagal, and A. Joshi. A secure infrastructure for service discovery and access in pervasive computing. *ACM Monet: Special Issue on Security in Mobile Computing Environments*, October 2003

[UBJ04] A. Uszok, J. M. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton and S. Aitken, Policy and Contract Management for Semantic Web Services. In *Proceedings of Semantic Web Services Symposium, AAAI 2004 Spring Symposium Series*, Stanford University, Stanford California.

[W3C04] OWL-S: Semantic Markup for Web Services, W3C Submission Member Submission, November 2004. http://www.w3.org/Submission/OWL-S