Semantic Web Framework and Meta-Control Model to Enforce Context-Sensitive Policies

Jinghai Rao and Norman Sadeh

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

Abstract. Enforcing rich policies in open environments will increasingly require the ability to dynamically identify external sources of information necessary to enforce different policies. In this paper, we introduce a semantic web framework and a meta-control model 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 relies on one or more software 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, 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 though the proposed framework extends to a number of other environments (e.g. virtual enterprises, coalition forces, homeland security). Preliminary empirical results appear rather promising.

1 Introduction

As Web applications aim for increasingly high levels of sophistication and automation, there will be a growing need for enforcing complex policies whose satisfaction is not tied to predefined sources of information. An example is enforcing context-sensitive security and privacy policies, whether in pervasive computing applications or in support of virtual enterprise scenarios, coalition force scenarios or interagency collaboration in a homeland security context.. 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, but, when she is not, this information can possibly be obtained via her cell phone operator);

– Available sources of information may not be known ahead of time (e.g. new location tracking functionality may be installed or the user may roam into a new area).

Enforcing context-sensitive policies in 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 trust management infrastructures proposed so far and calls for privacy and security enforcing mechanisms capable of operating external services.

We introduce a semantic web framework and a meta-control model 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. In this paper, we look more particularly at the issue of enforcing privacy and security policies in pervasive computing environments. In this context, the owner of information sources relies on one or more software agents for enforcing relevant policies in response to incoming requests. These agents implement meta-control strategies to interleave policy enforcement, semantic web reasoning and service discovery and access. This paper introduces one particular type of agent we refer to as Information Disclosure Agents (IDA), who 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 (see section 2 for more detail). The research reported here has been conducted in the context of *MyCampus*, a pervasive computing environment aimed at enhancing everyday campus life at Carnegie Mellon University [6, 11].

The work presented in this paper builds on concepts of decentralized trust management developed over the past decade (see [3] as well as more recent research such as [1, 2, 7]). 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 [6, 10]. 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, our meta-control architecture interleaves 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 a software agent architecture for enforcing privacy and security policies. Section 3 details the meta-control model based on query status information. Section 4 discusses our service discovery model. Section 5 presents our current implementation and discusses initial empirical results. Concluding remarks are provided in Section 6. Additional details on the work described in this short paper, including a detailed description of the operation of our meta-control architecture can be found in [10].

2 Overall Approach and Architecture

We consider an environment where sources of information are all modeled as services that can be automatically discovered based on rich ontology-based service pro-

files advertised in service directories. Each service is applied to policies, which are represented as rules. In this paper we focus on access control policies and obfuscation policies enforced by *Information Disclosure Agents* (IDA), though the framework we present could readily be used to enforce a variety of other policies.

An IDA receives requests for information or service access. In processing the requests, it is responsible for enforcing access control and obfuscation polices specified by its owner. As it processes requests, the agent records status information that helps it monitor its own progress in enforcing its policies and in obtaining the necessary information to satisfy the request. Based on this updated *query status information*, a meta-control module ("meta-controller") dynamically orchestrates the operations of modules at its disposal to process queries (Fig. 1). 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.

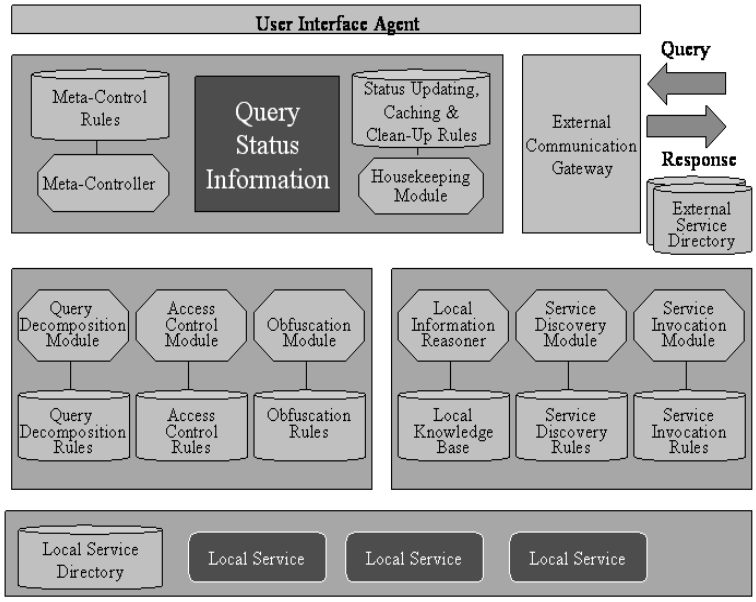

Fig. 1. Information Disclosure Agent: Overall Architecture

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 typically implemented as a single reasoning engine. The following provides a brief description of each of the modules orchestrated by an IDA's meta-controller:

- *Query Decomposition Module* takes as input a particular query and 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 query or subquery 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 sources. When this is the case, rather than immediately deciding whether or not to grant access to a query, the *Access Control Module* needs to request additional facts – also modeled as *Query Elements*.

- *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 requests for additional *Query Elements.*
- *Local Information Reasoner* corresponds to domain knowledge (facts and rules) known locally to the IDA.
- *Service Discovery Module* helps the IDA identify potential sources of information to complement its local knowledge. External services can be identified through external service directories (whether public or not), by communicating via the agent's *External Communication Gateway*. The service identification rules directly map information needs on pre-specified services. We currently assume that all service directories rely on OWL-S to advertise service profiles (see Section 4).
- *Service Invocation Module* allows the agent to invoke relevant services. It is important to note that, in our architecture, each service can have its own 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) and a potential source of meta-control knowledge (e.g. if a particular query element proves too difficult to locate, the user may be asked whether to stop looking).

3 Query Status Model

An 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 modeled as if-then clauses, with Left Hand Sides (LHSs) specifying their premises and Right Hand Sides (RHSs) their conclusions. LHS elements refer to query status information, while RHS elements contain facts that result in module activations. Query status information helps keep track of how far along the IDA is in obtaining the information required by each query and in enforcing relevant policies. Query status information in the LHS of meta-control rules is expressed according to 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, etc. All status information is annotated with time stamps. In other words, query status information includes:

- **Status predicates** to describe the status of a query or query element
- **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).

A list of query status predicates currently implemented can be found in [10]. In general, query status information is updated by asserting new facts (with old information being cleaned up by the IDA's housekeeping module). 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. As already mentioned earlier, this meta-control architecture can also be used to model the user as a module that can be consulted by the meta-controller, e.g. to ask for a particular piece of domain knowledge or to decide whether or not to abandon a particular course of action such as looking for an external service capable of providing a particular query element.

The following example illustrates a meta-control rule. This rule indicates the status change after a service is invoked successfully to tell the value of the required query element. Once the service response is received, the old status "waiting-for-service-response" is cleaned, and the new status "element-available" is generated. The rule, expressed in CLIPS [4], is of the form:

```
?x <- (triple "Status#predicate" ?s1 "waiting-for-service-response")
?y <- (triple "Query#queryId" ?s1 ?service)
(triple "Status#predicate" ?s2 "service-response-available")
(triple "Query#queryId" ?s2 ?result)
=>
(retract ?x)
(retract ?y)
(assert (triple "Status#predicate" ?newstatus "element-available))
(assert (triple "Query#queryId" ?newstatus ?result))
```

4 The Service Discovery Model

A central element of our method is the ability of IDA agents to dynamically identify sources of 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. .

Each service (or source of information) is described by a *ServiceProfile* in OWL-S [9]. In general, a *ServiceProfile* consists 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 [12]. 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 Y can be described as follows:

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

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

  <profile:hasPrecondition rdf:resource="&Process;#LocateInCompanyY"/>
  <profile:hasOutput rdf:resource="&Process;#RoomNoOutput"/>
</profileHierarchy:InformationService>
```

When invoking a service it has identified, an IDA may opt to provide upfront all the input parameters required by that service or it may withhold one or more of these parameters. The latter option forces the service to request the missing input parameters from the IDA, thereby enabling the IDA to more fully determine whether the invoked service meets its policies. This option is however more computation and communication intensive.

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 as output a contextual attribute 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 above has a precondition specifying that it is only available on company Y's premises.

5 Current Implementation: Evaluation and Discussion

Our policy enforcing agents are currently implemented in JESS, a high-performance rule-based engine in Java [5]. Domain knowledge, including service profiles, queries, access control policies and obfuscation policies are expressed in OWL [6]. As already indicated earlier, we use ROWL to define rules. XSLT transformations are used to translate OWL facts and ROWL rules into CLIPS, the rule language supported by JESS. 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 information exchange and without digital signatures. In the future, we plan to use SSL or some equivalent protocol for all information exchange. This will include signing all queries and responses.

We have evaluated our solution on an IBM laptop with a 1.80GHz Pentium M CPU and 1.50GB of RAM. The laptop was running Windows XP Professional OS, Java SDK 1.4.1 and Jess 7.0. As part of the evaluation, we implemented the example

introduced in Section 4 and 6, using a light-weight rule/fact set. The set included 22 rules and 178 facts and features a single semantic service directory with 50 services, each represented by 5 to 10 Jess rules. A breakdown of the CPU times required to process Bob's query is provided in the table below. For each module the table provides a cumulative CPU time, namely the sum of the CPU times of all invocations of that module in processing the query.

Module	CPU time in millisecond
Meta-Controller	28
Access-Controller	32
Local-KB	49
Service discovery / invocation	72
Total	181

While these results provide just one data point and only evaluate a subset of our functionality, they seem to suggest that our solution can be viewed as practical in at least some simple settings. It should be noted that our solution is not JESS-specific and could be implemented in other rule languages.

6 Concluding Remarks

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 software 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. These meta-control strategies can also be extended to treat the user as another source of information, e.g. to confirm whether a given fact holds or to provide meta-control guidance such as deciding when to abandon trying to determine whether a policy is satisfied.

The Information Disclosure Agent presented in this paper is just one instantiation of our more general concept of Policy Enforcing Agents (PEAs)[10]. 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 meta-control architecture and many of the same principles presented in this paper. In general, PEAs rely on a taxonomy of query information status predicates to monitor their own progress in processing incoming queries and enforcing relevant security and privacy policies. Preliminary evaluation of an early implementation of our framework seems encouraging. 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 further evaluating and refining 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 opportuni-

ties 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), breaking deadlocks [8], and integrating the user as a source of information.

References

[1] L. Bauer, M.A. Schneider and E.W. Felten. "A General and Flexible Access Control System for the Web", In Proceedings of the 11th USENIX Security Symposium, August 2002.

[2] 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, Carnegie Mellon University, February 2005.

[3] M. Blaze, J. Feigenbaum, an J. Lacy. "Decentralized Trust Management". Proc. IEEE Conference on Security and Privacy. Oakland, CA. May 1996.

[4] CLIPS. http://www.ghg.net/clips/CLIPS.html.

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

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

[7] 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, 2004

[8] 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 ISWC04, Hiroshima, Japan, November 2004.

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

[10] J. Rao and N.M. Sadeh. Interleaving Semantic Web Reasoning and Service Discovery to Enforce Context-Sensitive Security and Privacy Policies. Carnegie Mellon University Technical Report (CMU-ISRI-05-113), July 2005. http://www-2.cs.cmu.edu/~sadeh/Publications/More%20Complete%20List/techreport%20%20july%2027%202005.pdf

[11] N.M. Sadeh, F. Gandon, and Oh Byung Kwon. Ambient Intelligence: The MyCampus Experience. Carnegie Mellon University Technical Report (CMU-ISRI-05-123). June 2005.

[12] 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.

Acknowledgements

The work reported herein has been supported in part under DARPA contract F30602-02-2-0035 ("DAML initiative") and in part under ARO research grant D20D19-02-1-0389 ("Perpetually Available and Secure Information Systems") 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.