Declarative Policies for Describing Web Service Capabilities and Constraints

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

1 Overview

Though the description of capabilities and constraints for web services is an important problem, we believe that is it part of a much larger problem : controlling the behavior of autonomous entities in open, dynamic environments. This problem deals with not only with the specification of attributes (i.e. privacy restrictions, access control rules, communication requirements) that will enable interacting entities to behave appropriately, but also with the specification of all aspects of the behavior of entities (i.e. what entities can or must or may do under certain circumstances). Actually, the former specification is a subset of the latter. We believe that research into governing behavior of autonomous entities like agents and web services will provide suitable solutions to these kind of specifications.

We propose that behavior can be described using declarative policies that are based on deontic concepts including permissions, obligations, claims, prohibitions, and privileges. These policies will describe what the ideal behavior for an entity is in a certain context. For example, the constraint *You must use HTTP Authentication when accessing this service* can be modeled as appropriate behavior for an entity (agent, web service, human user) that wants to use a service. The entity is 'permitted' to access the service if it meets a certain condition i.e. uses HTTP authentication. It can be described as an access control policy for the service. However, these policy specifications should not only be able to represent security, but all aspects of behavior including privacy, management, conversation, etc. Another example is *You MAY use GZIP compression.* This can represented as a 'privilege' or a 'claim', but it again represents the ideal behavior of the entity. Negative modalities should also be possible. For example, *You SHOULD not use my contact details for marketing of services or products* 'prohibits' the entity (in this case a website) from performing a certain action and *Authentication is not required if a valid cookie is set* is a dispensation that frees the entity from the obligation of authenticating itself.

Even though these policies represent how the entity should ideally behave, whether the entity conforms to the policy depends either on the entity or the enforcement mechanism. In the case of web services, it is possible to include the enforcement mechanism into brokering services like the OWL-S MatchMaker or Virtual Machine, which act as a liaison between the interacting entities [1, 2]. However, in truly dynamic open environments, there will be peer-peer communication and entities will be responsible for their own behavior. So, along with enforcement, we also propose a more normative approach, where each entity is capable of reasoning over its own policies and goals, and the policies of the entities it

needs to interact with, in order to infer how it should behave. In order to meet this requirement, an entity should be able to understand the policies applicable to it. We propose that machine-understandable specification languages should be used to describe policies over shared ontologies. These policies should not only include norms of different kinds of behavior but also model the consequences of deviating from the policy in order to influence an entity's decision to adhere to the policy.

2 Our Current Work

In order to provide policy-based control over entities in dynamic environments, we have designed a policy specification language, Rei[1], which is currently expressed in OWL-Lite. Rei allows the specification of declarative policies over domain ontologies in RDF, DAML+OIL and OWL. As Rei is based in OWL, Rei policies can describe entities and the context at different levels of abstraction, from a specific instance to the most general class that subsumes it. Though represented in OWL-Lite, Rei includes logic-like variables giving it the flexibility to specify a range of relations that are not directly possible in OWL like role value maps. For example, it is possible to describe *uncle of* and *same group as* relations.

Rei permits policies to be specified as norms of behavior [3, 4]. Rei policies restrict the actions that an entity can/must perform on resources in the environment. We believe that most policies can be expressed in terms of what an entity can do (authorization) and what it should do (obligation) in terms of actions, services, conversations etc., making Rei capable of describing a large variety of policies ranging from security policies to conversation [5] and behavior policies.

Rei policies can be defined either over an individual actor, action and target or a class of actors, actions and target as defined by the associated conditions. The latter is useful in large scale distributed environments where stating individual permissions and obligations is time consuming. Rei provides a rule language for defining individual and group based policies. Group based policies can be defined using variables to express a set of constraints over the actor, target and action. There are three main sections of Rei specifications : the deontic notions of permissions, prohibitions, obligations and dispensations, speech acts for remote policy management including delegation, revocation, request, and cancel, and meta-policies for dynamic conflict detection and resolution. Rei also supports policy analysis tools that aid in the development of consistent policies.

As open, dynamic environments usually have overlapping policies of behavioral norms, constraints, and rules, the entities will be over-constrained. This means that they cannot always satisfy all of the policies, but deviating too much or too often has its consequences - loss of reputation, penalty clauses, imposition of sanctions, etc. Rei provide mechanisms for modeling these consequences so that autonomous entities can reason over them to decide whether or not to deviate from a certain policy. Towards this end, Rei allows 'sanctions', which describe the penalties of violating the policy, to be attached to prohibitions and obligations. A 'sanction' is capable of expressing common penalties of violating policies in human societies : retracting a permission, granting an additional obligation, or reducing a reputation measure.

We have developed a policy engine that reasons over Rei policies, domain information and the context to answer queries about the current permissions and obligations of entities in the environment.

[1] http://rei.umbc.edu/

Application	Domain Knowledge	Behavior
Coordinating access in supply chain management [6, 7]	X.509 certificates, company ontology describing users, roles, projects, etc.	Authorization
Authorization policies in pervasive computing environments [8–10]	X.509 certificates, XML service descriptions, university ontology describing users, labs, advisor, and resources	Authorization
Collaboration in Multi-Agent Systems [11]	X.509 certificates , descriptions of crisis, teams, users, and agencies	Team formation, Collaboration, Information Filtering
Security, Privacy and Confidentiality in semantic web services [12, 1, 2]	OWL-S, Friend of a Friend (FOAF), university ontology	Authorization, Privacy, and Confidentiality
Privacy and trust on the Internet [13]	P3P, FOAF, website evaluation ontology	Privacy
Enforcing domain policies on handheld devices in pervasive computing environments [14]	NIST ontology for enforcement, university ontology for students, faculty and computing resources	Control of Handheld device
Security for Task Computing	Fujitsu company and service ontologies, OWL-S service descriptions	Authorization

Table 1. Rei has been used in a number of prototype applications at UMBC and several companies to define and reason over policies for authorization, privacy and trust

It is able to answer several types of queries including, (i) Does X have the permission to perform Y on resource Z, (ii) What are the current obligations of X, (iii) What actions can X perform on resource Z, (iv) What are all of X's permissions in the current policy domain, and (v) Under what conditions does X have the permission to perform Y on resource Z. While answering these queries, the Rei engine takes into account all applicable speech acts and policies and tries to resolve detected conflicts using defined meta policies. The Rei engine can be used to check policy compliance and its results can be used either by the enforcement mechanism or by norm-governed entities.

3 Case Studies

As Rei is capable of expressing different kinds of policies over domain specific ontologies, Rei has been used in several open, dynamic distributed applications to provide different kinds of behavior. Table 1 describes some of these applications.

4 Use Case

Based on our approach to controlling behavior using declarative policies, the use case can be described as follows :

– A Web service, WS, wishes to stipulate that clients are required support a reliable messaging protocol and encrypt a specific header with WS-Security using a X.509 or user name security token

in order to send an acceptable request message : WS defines this requirement as an Rei access control policy over a set of conditions. Only when the conditions are satisfied, will the requesting client be 'permitted' to access the service. Information about what it means to be a 'reliable messaging protocol' or 'encrypt specific header with WS-Security' will be expressed as domain ontologies.
- Furthermore, the service has a P3P policy associated with its operations : This policy can either be expressed in P3P (Rei is able to reason over P3P policies as well [13]) or can be described as a privacy policy using the P3P RDF ontology.
- Let us assume a client, C, has a policy as well, a Rei privacy policy specified over P3P RDF specifications and user context as described in [13].
- If C and WS are to interact, one possible procedure is :
  - C contacts WS
  - WS sends C its access control policy and P3P policy
  - C sends WS its privacy policy and a description of its privileges, and permissions
  - C verifies that WS meets its privacy requirements by checking the P3P policy against its own privacy requirements.
  - WS verifies that the C has the permission or privilege to support a reliable messaging protocol and encrypt a specific header with WS-Security using a X.509 or user name security token by checking C's privileges and permissions
- The Rei policy engine currently supports the policy compliance testing done in this usecase.
- A problem with this approach, is that each party has to disclose its policy. Another approach is to use a trusted third part for checking policy compliance or use a negotiation protocol, where every entity reveals a little more during every message exchange.

5 Open Issues

Some interesting open research issues include :

- Can we find the right balance between having a policy language or framework that is (i) powerful and expressive enough to handle a wide range of uses, (ii) efficient enough for real applications and (iii) reasonably easy for real developers to learn and use.
- Should the paradigm be dominated by a rule based approach (e.g., using RuleML or SWRL) or an ontology approach (e.g., using OWL), a hybrid of the two, or something else altogether.
- What are appropriate mechanisms for enforcement? How can we make them easy for developers to integrate into their systems?
- Though its specifications states that an entity behaves in a certain way, how can we guarantee this behavior ? For example, a website states that it will not provide your contact information and if it does, it is obliged to inform you. How can this be guaranteed ? Reputation schemes, third party verifiers ?
- What methodologies, metrics and tools can be developed to support "policy engineering" – the specification, implementation, debugging, documentation, maintenance and dynamic composition of policies.
- Should policy decisions be justified ? If yes, how ?
- What kinds of policies should be kept private ? How can policies be kept private but still be understood and adhered to ?

References

1. Kagal, L., Paolucci, M., Srinivasan, N., Denker, G., Finin, T., Sycara, K.: Authorization and Privacy in Semantic Web Services. In: First International Semantic Web Services Symposium at the AAAI Spring Symposium, Stanford. (2004)
2. Kagal, L., Paolucci, M., Srinivasan, N., Denker, G., Finin, T., Sycara, K.: Authorization and Privacy in Semantic Web Services. In: IEEE Intelligent Systems (Special Issue on Semantic Web Services). (2004)
3. 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)
4. Kagal, L., Finin, T., Joshi, A.: A Policy Based Approach to Security for the Semantic Web. In: Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003. (2003)
5. Kagal, L., Finin, T.: Modeling Conversation Policies using Permissions and Obligations. In: AAMAS 2004 Workshop on Agent Communication (AC2004). (2004)
6. Kagal, L., Finin, T., Peng, Y.: A Delegation Based Model for Distributed Trust. In: Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents, International Joint Conferences on Artificial Intelligence. (2001)
7. Kagal, L., Finin, T., Peng, Y.: A Framework for Distributed Trust Management. In: IJCAI-01 Workshop on Autonomy, Delegation and Control. (2001)
8. Kagal, L., Finin, T., Joshi, A.: Trust based security for pervasive computing enviroments. In: IEEE Communications, December 2001. (2001)
9. Kagal, L., Korolev, V., Chen, H., Joshi, A., Finin, T.: Centaurus : A Framework for Intelligent Services in a Mobile Environment. In: International Workshop of Smart Appliances and Wearable Computing at the 21st International Conference of Distributed Computing Systems. (2001)
10. 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)
11. Cornwell, M., Just, J., Kagal, L., Finin, T.: A Policy Based Collaboration Infrastructure for P2P Networking. In: Twelfth International Conference on Telecommunication Systems, Modeling and Analysis, July 2004. (2004)
12. Denker, G., Kagal, L., Finin, T., Paolucci, M., Sycara, K.: Security for DAML Web Services: Annotation and Matchmaking. In: Second Int. Semantic Web Conference (ISWC2003), Sanibel Island FL, October 2003. (2003)
13. Kolari, P., Kagal, L., Joshi, A., Finin, T.: Enhacing P3P Framework with Policies and Trust. UMBC Technical Report and under review (2004)
14. Patwardhan, A., Korolev, V., Kagal, L., Joshi, A.: Enforcing policies in Pervasive Environments. In: International Conference on Mobile and Ubiquitous Systems: Networking and Services. (2004)