Compliance Checking of Privacy Policies for Semantic Web Services

Grit Denker[1] and Son Nguyen[2]

[1]SRI International, Menlo Park, CA 94025, Grit.Denker@sri.com
[2]San Jose State University, San Jose, CA 95192, sonkylenguyen@yahoo.com

Abstract

In order to reach the full potential of Web-based applications one needs to address the privacy concerns of customers. We propose a framework for specification and compliance checking of privacy policies in the context of Semantic Web Service applications. We defined an ontology for privacy policies and implemented a compliance checking algorithm.

Keywords: Semantic Web, Semantic Web service, ontology, privacy, policy, compliance.

1. Introduction

Web Service (WS) technology is now widely used in B2B applications and it is also becoming more widespread in B2C transactions. As a consequence, privacy issues are more important and need to be addressed to assure customers and clients of the appropriate use of their confidential data. The possibility of privacy violation, whether it is the unauthorized collection of private data or its distribution without requesting consent from the owner of the data, creates fear among Web users [CRA02]. Hence, there is a need for privacy policies that define rules and concepts to protect private personal and enterprise information. A solid framework that handles privacy policies and guarantees service compliance with policies will increase the trust of users in this technology and lower the adoption barrier for automated service applications.

WS are a solution to integrate distributed Web-based applications even though if they have been developed in different programming or on vendor-specific platforms. Semantic Web services (SWSs) promise to provide solutions to the challenges associated with automated discovery, dynamic composition, enactment, and other tasks associated with managing and using service-based systems. In order to realize interoperable, automated, and trustworthy distributed service applications, it is necessary to investigate how privacy policies can be integrated into a SWS approach.

In this paper we propose an ontology-based approach to privacy policy specification in SWS applications and checking compliance using matching technology. In Section 2, we start with a brief overview of existing policy languages and identify what concerns have not been adequately addressed in those languages. These considerations influenced the design of our privacy ontology that is presented in Section 3. We illustrate the use of our ontologies with the help of client and service privacy policies. In Section 4 we briefly present the communication protocol we implemented to exchange privacy policies between client and service. We also introduce the reasoning algorithm that was designed and implemented to determine the compatibility of client and service privacy policies. The algorithm provides feedback to the user to help make a decision whether a service's privacy policy is acceptable for a transaction. We conclude with a brief summary and ideas for future work in Section 6.

2. Machine-readable Policy Languages

Our main focus is on semantic specification of service privacy policies that will be used in automated reasoning. Given the context of SWS, we have looked into the existing XML-based policies languages. We briefly address three of the major languages, namely Platform for Privacy Preferences Project (P3P), KAoS, and Rei. There exist other XML-based languages such as Enterprise Privacy Authorization Language (EPAL) or policy languages such as Ponder for which many of the remarks below also apply.

The XML-based language P3P (Platform for Privacy Preferences) was developed by the World Wide Web Consortium (W3C) to create a standard way for Web sites to define and communicate privacy polices [Cra02]. A P3P-enabled Web browser allows a user to enable or disable a set of preference rules and match these rules are against the privacy policy of visited Web sites. P3P is based on XML and does not

support semantically expressive policy descriptions. Rei [TBJ+03], an RDF-based language, can be used to describe different types of policies, including security policies. Rei describes policies with deontic concepts such as right, prohibition, obligation and dispensation. A Prolog-based engine for Rei reasons about polices. Currently, Rei is missing the privacy specific concepts (e.g., disclosure and storage among others) that we propose in our ontology. KAoS [UBJ+03] uses OWL to specify policy ontologies. KAoS is more than just a language. It provides ways to define and manage domain and policy services such as resolving conflicts among polices and make sure polices specifications are enforced. Similarly to Rei, KAoS embraces the concepts of authorization and obligations but does not privacy-specific concepts.

3. An OWL Privacy Schema

An example of a privacy policy is to assign a company the right to access the client's private information with an obligation of not giving this information to a third party. In the SWS world, each intelligent agent should carry its own privacy policy. Similarly, each Web service has privacy policies attached to it. Exchanging privacy policies between agent and service can be a first step in an interaction to determine whether the agent meets the privacy set forth by the service and vice a versa. In the following we provide more examples of privacy policies to give an idea of what kind of policy we want to be able to specify with our ontology.

3.1. Privacy Policy Examples

SWS privacy policies are specified on the client side and the server side. Sometimes a server can also act as a client. Hence, SWS privacy policies can also be applied within a distributed environment where server-server communication is required. Therefore, we use in the following the notions "provider" and "requester" instead of "server" and "client", as a service might become client of another service. Below we list example of privacy policies. The term "private information" refers to personal or organizational information such as name, organization, address, phone number, health record, credit cards, hobbies, interests, etc. The examples in the table are such that the requester and the provider policies match.

Requester Privacy Policy/Provider Privacy Policy:

1. Allow access and store my private information / Will collect privacy information from clients
2. Prohibit others to collect my private information / Will not collect any private information
3. Do not disclose my private information to 3rd parties and do not collect clickstream info / Will collect private information but will not distribute it to 3rd party and will not collect clickstream information
4. My private information must be kept secre and the integrity of private data must be protected / Support storage encryption and signature
5. Allow to keep private information within a certain period of time / Specify data retention time

3.2. Privacy Ontology

Our goal is to develop a generic, simple and easy-to-use ontology for expressing privacy policies as well as a protocol to support matching of privacy policies. The protocol will be used to assure matching of the policies (see Section 4), and, thus, provides a first step toward policy enforcement.

We use the Semantic Web (SW) language OWL, a W3C Standard (http://www.w3.org/2001/sw/WebOnt), to define our privacy ontology. OWL provides semantically rich vocabulary and its semantics that can be used for reasoning over privacy policies. Moreover, this approach does also allow us to make use of the existing inference engines for OWL such as Jena [Jena2].

3.3. An OWL Privacy Policy

The main classes and properties of our privacy policy formalize policy rules and actions. The OWL privacy ontology and example policies can be found at http://www.csl.sri.com/users/denker/owl-sec/owl-sec/privcacy. In the following we give details about the **Classes** and the *properties* defined in our privacy ontololgy.

Policy Rules

An **Entity** may have a **Policy** as indicated by the *hasPolicy* property. A **Policy** can have one or more **Rules,** as indicated by *hasRule* property. A **Rule** applies to one or more actions (*onAction*) and a specific type of privacy information (*onResource*). The range of the *onResource* property is any OWL class. By default, if no resource is specified, then the rule applies to all types of resources. The range of the rule property "*onResource*" is "owl:Class". This allows maximum flexibility for specifying privacy policies over any kind of data, because the user can decide which OWL class is the one that is protected by a policy rule.

We defined **Authorization**, **Capability**, and **Obligation** as **Rule** subclasses. **Authorization** and

Obligation specify rules about actions that are allowed or required for the other party, whereas **Capability** specifies what the advertising party is able to perform. Authorization and Capability classes have each two subclasses specifying negative or positive policies (e.g., **NegativeCapability**). For example, a policy that allows a provider to collect the requester's preferences and clickstream information as long as this information is kept locally is defined as a positive authorization policy. A provider advertising its capability to send and receive encrypted data uses a positive capability rule, whereas the inability to but to send or receive signed data is formalized as a negative capability rule. Requiring a provider to provide data encryption when storing sensitive data can be defined as an obligation rule. The obligation class is not subclassed into negative and positive obligation since this is not meaningful.

There are two subclasses of the positive capability class, namely **PositiveIntentCapability** and **NegativeIntentCapability**. The former class is used to define rules about actions that the advertising party intends to perform, whereas the latter one is used to define rules about actions that the current party does not intent to perform though it has the capability to do so. For example, a party may be capable of encrypting data in transmission but does not do so by default. Negative intent capabilities could be turned into obligation during a negotiation process.

Actions and Protected Data

We defined a policy rule to be specified for a certain action (i.e., "*onAction*" property in policy ontology) and a certain resource (i.e., "*onResource*" property in policy ontology). The action ontology describes different types of actions that can in the range of the *onAction* property.

In the following we describe the **Subclasses** of the **Action** class. **DataCollection**, **Disclosure**, **Storage**, and **Transmission** are subclasses of **Action**. Some of the action classes are further refined into subclasses. For example, **Local**, **Third Party**, and **Forum** are subclassed from the **Disclosure** class and are used in rules to further specify what kind of disclosure is allowed or not. For example, the affiliation subclass is used for rules that allow the disclosure of sensitive information to people, companies, or organizations that have a partnership with the entity. The **Storage** class is subclassed into **Storage Encryption** and **Storage Plaintext.** The **Transmission** class has various subclasses for encrypted, signed or plaintext sending and receiving.

4. Checking Policy Compatibility

Privacy policy rules need to be enforced by both communicating parties. We implemented a protocol for exchanging policies and an algorithm for checking policy compliance. The protocol is a simple requester-initiated protocol that sends the requester policy to the provider. The provider replies by sending its policy to the requester. Both, requester and provider run the same algorithm to check compliance of the policies. If the policies of the requester and the provider do not match, that is, one or more requirements of either party are not fulfilled by the other party, no communication will be established. If the policies match, then in principle a communication between the two communicating parties can be established. There is still more to do to enforce the policies, For example, required encryption mechanisms have to be put into place. This is not within the scope of this paper. In this paper we focus on the definition of privacy policies and provide mechanisms that enforce matching of policies between communicating parties.

4.1. Pairs of Rules

Matching of policies is done rule by rule. A rule has a certain type (e.g., authorization or obligation or capability), a set of actions that it governs, and possibly the type of data that is protected by the rule. For each rule of one party the matching algorithm tries to find a match in the set of rules in the other party. Not each pair of rules is relevant for the matching. For example, if one party states a negative authorization about storing private information and the other party states a negative capability of storing private information, then those two rules are relevant for the matching algorithm. On the other hand, if one party has a positive authorization rule, then it is not crucial to know whether the other party intends to make use of the positive authorization by defining a corresponding positive intent. Thus, in general a party is concerned to check those rules of its privacy policy that restrict the actions of the other party (negative authorization) or that require the other party to perform certain actions (obligation). For example, for the policies in Section 3.1, the following are rule pairs that are *possibly* relevant for checking compliance. We use "possibly relevant", because only if the rules address the same action and data, they are relevant for deciding compliance.

Requester Privacy Policy / Provider Privacy Policy

1. Prohibits other party to disclose private data to a 3rd party (neg authorization) /
    a. Will not disclose private data to a 3rd party (neg intent capabitility)

   b. Will collect sensitive data from client and disclose information locall (pos intent cabability)
   c. Cannot encrypt collected information (neg capability)
2. Service provider must encrypt private data when it is stored (obligation)
   a. Cannot encrypt collected information (neg capability)
   b. Will not disclose private data to 3rd party (neg intent capability)

In general, rule types that might constitute a possible match or conflict are called "*rule pairs.*"
Rule pairs that constitute a possible conflict are "neg authorization/pos capability", "neg authorization/pos intent capability", "obligation/neg capability", and "obligation/neg intent capability". Similarly, there are other rule pairs that might constitute a match such as "neg authorization/neg capability" among others.

If a rule pair is found, one still needs to check on the details of the actions and protected resources. If two rules in a rule pair address the same action and data (or, more generally, if the actions and protected resources in the provider rule are subsumed by the actions in the requester rule), then one can make the decision whether there is a conflict of policies. Therefore, the compliance checking algorithm iterates over all rules, trying to find rule pairs, and then, for each rule pair checks subsumption of the actions and resources protected.

4.2. Pseudo Code of Policy Compliance Algorithm

In our implementation we have used the Jena2 API from Hewlett Parker Laboratory [Jena2] to support OWL parsing and reasoning. The Jena OWL reasoner is an instance-based reasoner. For example, with Jena, we can infer if an instance A has the same class or subclass of an instance B. "*Model*, *Reasoner* and *InfModel"* are three important components that Jena uses for all inference. A *Model* can be built from an OWL document, a *Reasoner* is built based on a Model, and an *InfModel* can be constructed for reasoning based on *Model* and *Reasoner*.

The following steps describe the high level privacy policy compliance matching algorithm.

1. Read both client and server privacy policy. For each privacy policy: (a) Construct inference models for reasoning based on the privacy ontology, and (b) recursively parse OWL privacy information from the policy. The information is saved in the memory for future access.
2. Get all possible rule pairs of the two polices. For each possible rule pair: (a) Match types of protected or sensitive resources, and (b) match actions.

In general, the result of matching can be classified into three categories: (1) number of unsupported rules, (2) number of mismatched sensitive data, and (3) number of unsupported actions. An unsupported rule of one policy is a rule for which no pairing rule is found in the other policy. For a given rule pair, a mismatch sensitive data is data that is not supported by the other rule, and an unsupported action is an action that is not supported by the other rule.

5. Concluding Remarks

We proposed a privacy ontology that can be used to specify privacy policies that are useful in client-server applications. We designed and implemented an algorithm that checks the consistency between policies. This is a first step towards automated Web service applications. Our framework can help to increase the trust of clients in Web Services. As a next step we will investigate how to extend our framework to negotiation. The results of the reasoning algorithm can be exploited to suggest changes to policies that would yield compliant policies.

6. References

[1] [Cra02] L. F. Cranor. Web Privacy with P3P. Copyright 2002 AT&T. Published by O'Reilly & Associates, Inc., Sebastopol, CA 95472.

[2] [TBJ+03] G. Tonti and J. Bradshaw and R. Jeffers and R Montanari and N. Suri and A. Uszok. Semantic Web Languages for Policy Representation and Reasoning: A Comparision of KAoS, Rei, and Ponder. In Proc. 2nd Intern. Semantic Web Conference (ISWC 2003). Springer.

[3] [UBJ+03] A. Uszok and J. Bradshaw and J. Jeffers and N. SUri and P. Hayes and M. Breedy and L. Bunch and M. Johnson and S. Kulkarni and J. Lott. KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement. In Proc. of the IEEE Workshop on Policy 2003, IEEE Press.

[4] [Jena2] Hewlett Parker Lab. Jena2: A Semantic Web Framework. http://jena.sourceforge.net/