26

Security and Privacy on the Semantic Web

Daniel Olmedilla

L3S Research Center and University of Hannover
Germany

Summary. The semantic Web aims to enable sophisticated and autonomic machine-to-machine interactions without human intervention, by providing machines not only with data but also with its meaning (semantics). In this setting, traditional security mechanisms are not suitable anymore. For example, identity-based access control assumes that parties are known in advance. Then, a machine first determines the identity of the requester in order to either grant or deny access, depending on its associated information (e.g., by looking up its set of permissions). In the semantic Web, any two strangers can interact with each other automatically and therefore this assumption does not hold. Hence, a semantically enriched process is required in order to regulate automatic access to sensitive information. Policy-based access control provides sophisticated means to support the protection of sensitive resources and information disclosure. This chapter provides an introduction to policy-based security and privacy protection by analyzing several existing policy languages. Furthermore, it shows how these languages can be used in a number of semantic Web scenarios.

26.1 Introduction

Information provided in the current Web is mainly human oriented. For example, HTML pages are human understandable but a computer is not able to understand the content and extract the concepts represented there, that is, the meaning of the data. The semantic Web [1] is a distributed environment in which information is self-describable by means of well-defined semantics, that is, machine understandable, thus providing interoperability (e.g., in e-commerce) and automation (e.g., in searching). In such an environment, entities which have not had any previous interaction may now be able to automatically interact with each other. For example, imagine an agent planning a trip for a user. It needs to search for and book a plane and a hotel taking into account the user's schedule. When the user's agent contacts a hotel's website, the latter needs to inform the former that it requires a credit card in order to confirm a reservation. However, the user may probably want to restrict the conditions under which her agent automatically discloses her personal information. Due to such exchange of conditions and personal information, as well as its automation, security and privacy become yet more relevant and traditional approaches are not suitable anymore. On the one hand, unilateral access control is now replaced by bilateral protection (e.g., not only does the website state the conditions to be satisfied in order to reserve a room but the user agent may also communicate conditions under which a credit card can be disclosed). On the other hand, identity-based access control cannot be applied anymore since users are not known in advance. Instead, entities' properties (e.g., the user's credit card or whether a user is a student) play a central role. Both these properties and conditions stating the requirements to be fulfilled by the other party, must be described in a machine-understandable language with well-defined semantics allowing other entities to process them. Systems semantically annotated with policies enhance their authorization process allowing, among others, to regulate information disclosure (privacy policies), to control access to resources (security policies), and to estimate trust based on parties' properties (trust management policies) [2].

Distributed access control has addressed some of these issues, though not solved them yet. Examples like KeyNote [3] or PolicyMaker [4], which are described in Chap. 8, provide a separation between enforcement and decision mechanisms by means of policies. However, policies are bound to public keys (identities) and are not expressive enough to deal with semantic Web scenarios. Role-based access control (see Chap. 5) also does not meet semantic Web requirements since it is difficult to assign roles to users which are not known in advance. Regarding user's privacy protection, the platform for privacy preferences (P3P), which is described in Chap. 25, provides a standard vocabulary to describe webserver policies. However, it is not expressive enough (it is a schema, not a language, and only describes the purpose for the gathered data) and it does not allow for enforcement mechanisms. On the other hand, a wide variety of policy languages have been developed to date [5, 6, 7, 8, 9], addressing the general requirements for a semantic Web policy language: expressiveness, simplicity, enforceability, scalability, and analyzability [10]. These policies can be exchanged between entities on the semantic Web and therefore they are described using languages with well-founded semantics.

The policy languages listed above differ in expressivity, the kind of reasoning required, features and implementations provided, etc. For the sake of simplicity, they are divided according to their protocol for policy exchange between parties, depending on the sensitivity of policies. On the one hand, assuming that all policies are public and accessible (typical situation in many multi-agent systems), the process of evaluating whether two policies from two different entities are compatible or not consists of gathering the relevant policies (and possibly relevant credentials) from the entities involved and checking whether they *match* (e.g., [11]). On the other hand, if policies may be private (the typical situation for business rules [12]), it implies that not all policies are known in advance but they may be disclosed at a later stage. Therefore, a *negotiation* protocol in which security and trust is iteratively established is required [13].

However, specifying policies is as difficult as writing imperative code, getting a policy right is as hard as getting a piece of software correct, and maintaining a large number of them is even harder. Fortunately, ontologies and policy reasoning may help users and administrators with the specification, conflict detection and resolution of such policies [5, 14].

This chapter first describes how policies are exchanged and how they interact among parties on the semantic Web, with a brief description of the main semantic Web policy languages and how ontologies may be used in policy specification, conflict detection and validation. Then, some examples of application scenarios are presented, where policy-based security and privacy are used, followed by some important open research issues. This chapter focuses only on policy-based security, privacy and trust on the semantic Web and does not deal with approaches based on individual trust ratings and propagation through a web of trust providing a means to rate unknown sources [15, 16, 17].

26.2 Policy-Based Interaction and Evaluation

Policies allow for security and privacy descriptions in a machine-understandable way. More specifically, service or information providers may use security policies to control access to resources by describing the conditions a requester must fulfil (e.g., a requester to resource A must belong to institution B and prove it by means of a credential). At the same time, service or information consumers may regulate the information they are willing to disclose by protecting it with privacy policies (e.g., an entity is willing to disclose its employee card credential only to the webserver of its employer). Given two sets of policies, an engine may check whether they are compatible, that is, whether they match. The complexity of this process varies depending on the sensitivity of policies (and the expressivity of the policies). If all policies are public at both sides (the typical situation in many multi-agent systems), provider and requester, the requester may initially already provide the relevant policies together with the request and the evaluation process can be performed in a one-step evaluation by the provider policy engine (or an external trusted matchmaker) and return a final decision. Otherwise, if policies may be private, as it is, for example, typically the case for sensitive business rules, this process may consist of several steps of negotiation in which new policies and credentials are disclosed at each step, advancing after each iteration towards a common agreement. In this section we give an overview of both types of languages. The main features of these languages are shown in Table 26.1. Additionally, we use the running policy “only employees of institution XYZ may retrieve a file” to illustrate an example of each language.

26.2.1 One-Step Policy Evaluation

Assuming that policies are publicly disclosable, there is no reason why a requester should not disclose its relevant applicable policies together with its request. This way, the provider's policy engine (or a trusted external matchmaker if the provider does not have one) has all the information needed to make an authorisation decision. The KAOS and REI frameworks, specially designed using semantic Web features and constructs, fall within this category of policy languages, those which do not allow policies themselves to be protected.

Table 26.1. Comparison of KAOS, REI, PeerTrust and Protune[1]

Policy language	Authorization protocol	Reasoning paradigm	Conflict detection	Meta-policies	Loop detection
KAOS	One-step	DL	Static detection and resolution		
REI	One-step	DL + variables	Dinamyc detection and resolution	Used for conflict resolution	
PeerTrust	Negotiation	LP + ontologies			Distributed tabling
Protune	Negotiation	LP + ontologies		Used for driving decisions	

KAOS Policy and Domain Services

KAOS services [5, 18] provide a framework for the specification, management, conflict resolution and enforcement of policies, allowing for distributed policy interaction and support for dynamic policy changes. It uses OWL [19] ontologies (defining, e.g., actors, groups and actions) to describe the policies and the application context, and provides administration tools (KAOS administration tool - KPAT) to help administrators to write down their policies and hide the complexity of using OWL directly. A policy in KAOS may be a positive (respectively negative) authorization, i.e., constraints that permit (respectively forbid) the execution of an action, or a positive (respectively negative) obligation, i.e., constraints that require an action to be executed (respectively waive the actor from having to execute it). A policy is then represented as an instance of the appropriate policy type, associating values to its properties, and giving restrictions on such properties (Fig. 26.1 sketches part of a KAOS policy).

[1] DL refers to description logic while LP stands for logic programming

KAOS benefits from the OWL representation and description-logic-based subsumption mechanisms [20]. Thus, it allows one to, for example, obtain all known subclasses or instances of a class within a given range (used during policy specification to help users choosing only valid classes or instances) or detect policy conflicts (by checking the disjointness of subclasses of the action class controlled by policies). KAOS is able to detect three types of conflicts, based on the types of policies that are allowed in the framework: positive vs. negative authorization (a policy allows access and but another denies it), positive versus negative obligation (a policy obliges to execute an action while another dispensates from such obligation) and positive obligation versus negative authorization (a policy obliges to execute an action but another denies authorization for such execution). KAOS resolves such conflicts (also called harmonization) based on assigning preferences to policies and resolving in favor of the policies with higher priority (Sect. 26.2.3 will extend on this).

Finally, KAOS assumes a default authorization mechanism in case no policy applies to a request. It can be either “permit all actions not explicitly forbidden” or “forbid all actions not explicitly authorized”.

```
<owl:Class rdf:ID="RetrieveFileAction">
  <owl:intersectionOf>
    <owl:Class rdf:about="#AccessAction"/>
      <owl:Class>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#performedBy"/>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:oneOf rdf:parseType="Collection">
                <owl:Thing rdf:about="#EmployeeInstitutionXYZ"/>
              </owl:oneOf>
            </owl:Class>
          </owl:someValuesFrom>
        </owl:Restriction>
      </owl:Class>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>

<policy:PosAuthorizationPolicy rdf:ID="PolicyRetrieveFileAction">
  <policy:controls rdf:resource="#RetrieveFileAction"/>
  <policy:hasPriority>1</policy:hasPriority>
</policy:PosAuthorizationPolicy>
```

```
<policy:Policy rdf:ID=”RetrieveFilePolicy”>
  <policy:grants rdf:resource=”#Perm_Employee_XYZ”>
</policy:Policy>

<policy:Granting rdf:ID=#Perm_Employee_XYZ”>
  <policy:to rdf:resource=”#PersonVar”>
  <policy:deontic rdf:resource=”Perm_Retrieve_File”>
</policy:Granting>

<deontic:Permission rdf:ID=”Perm_Retrieve_File”>
  <deontic:actor rdf:resource=”#PersonVar”>
  <deontic:action rdf:resource=”&action;RetrieveFile”>
  <deontic:constraint rdf:resource=”#IsEmployeeXYZ”>
</deontic:Permission>

<constraint:SimpleConstraint rdf:ID=”IsEmployeeXYZ”>
  <constraint:subject rdf:resource=”#PersonVar”>
  <constraint:predicate rdf:resource=”&emp;affiliation”>
  <constraint:object rdf:resource=”&emp;XYZ”>
</constraint:SimpleConstraint>
```

Fig. 26.1. Example of KAOS (left) and REI (right) policies

REI

REI 2.0 [21, 11] expresses policies according to what entities can or cannot do and what they should or should not do. They define an independent ontology which includes the concepts for permissions, obligations, actions, etc. Additionally, as in KAOS, they allow the import of domain-dependent ontologies (including domain-dependent classes and properties). REI 2.0 is represented in OWL-Lite and includes logic-like variables in order to specify a range of relations.

REI policies (see Fig. 26.1 for an example) are described in terms of deontic concepts: permissions, prohibitions, obligations and dispensations, equivalently to the positive/negative authorizations and positive/negative obligations of KAOS. In addition, REI provides a specification of speech acts for the dynamic exchange of rights and obligations between entities: delegation (of a right), revocation (of a previously delegated right), request (for action execution or delegation) and cancel (of a previous request).

As in the KAOS framework, REI policies may conflict with each other (right versus prohibition or obligation versus dispensation). REI provides mechanisms for conflict detection and constructs to resolve them, namely, overriding policies (similar to the prioritization in KAOS) and definition at the meta-level of the global modality (positive or negative) that holds (see Sect. 26.2.3 for more details).

26.2.2 Policy-Driven Negotiations

In the approaches presented previously, policies are assumed to be publicly disclosable. This is true for many scenarios but there exist other scenarios where it may not hold. For example, imagine a hospital revealing to everyone that, in order to receive Alice's medical report, the requester needs an authorization from Alice's psychiatrist. Another example: imagine Tom wants to share his holiday pictures online only with his friends. If he states publicly that policy and Jessica is denied access, she may get angry because of Tom not considering her as a friend. Moreover, policy protection becomes even more important when policies protects sensitive business rules.

These scenarios require the possibility to protect policies (policies protecting policies) and the process of finding a match between requester and provider becomes more complex, since not all relevant policies may be available at the time. Therefore, this process may consist of several steps of negotiation, by disclosing new policies and credentials at each step, and therefore advancing after each iteration towards a common agreement [13]. For example, suppose Alice requests access to a resource at e-shop. Alice is told that she must provide her credit card to be granted access. However, Alice does not want to disclose her credit card just to anyone and she communicates to the e-shop that, before it gets her credit card, it should provide its Better Business Bureau certification. Once e-shop discloses it, Alice's policy is fulfilled and she provides the credit card, thus fulfilling e-shop's policy and receiving access to the requested resource (see Fig. 26.2).

Below, the two most recent languages for policy-driven negotiation are presented. They are also specially designed for the semantic Web. However, we refer the interested reader to other languages for policy-based negotiations [22, 23, 24], which may be applied to the semantic Web.

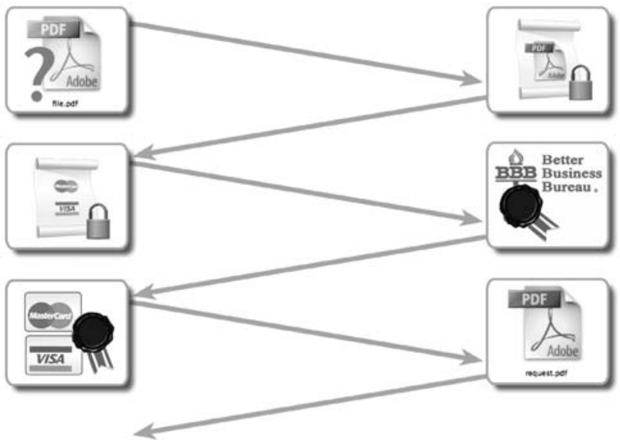


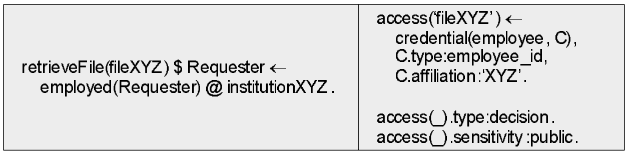
Fig. 26.2. Policy-driven negotiation between Alice and e-shop

PeerTrust

PeerTrust [7] builds upon previous work on policy-based access control and release for the Web and implements automated trust negotiation for such a dynamic environment.

PeerTrust's language is based on first-order Horn rules (definite Horn clauses), i.e., rules of the form "$lit_0 \leftarrow lit_1, \ldots, lit_n$" where each lit_i is a positive literal $P_j(t_1, \ldots, t_n)$, P_j is a predicate symbol, and the t_i are the arguments of this predicate. Each t_i is a term, i.e., a function symbol and its arguments, which are themselves terms. The head of a rule is lit_0, and its body is the set of lit_i. The body of a rule can be empty.

Definite Horn clauses can be easily extended to include negation as failure, restricted versions of classical negation, and additional constraint-handling capabilities such as those used in constraint logic programming. Although all of these features can be useful in trust negotiation, here we only describe other, more unusual, required language extensions. Additionally, PeerTrust allows the import of RDF-based meta-data, therefore allowing the use of ontologies within policy descriptions.



```
retrieveFile(fileXYZ) $ Requester ←
   employed(Requester) @ institutionXYZ.
```

```
access('fileXYZ') ←
   credential(employee, C),
   C.type:employee_id,
   C.affiliation:'XYZ'.

access(_).type:decision.
access(_).sensitivity:public.
```

Fig. 26.3. Example of PeerTrust (left) and Protune (right) policies

References to Other Peers PeerTrust's ability to reason about statements made by other peers is central to trust negotiation. To express delegation of evaluation to another peer, each literal lit_i is extended with an additional *Authority* argument, that is

lit_i @ Authority

where *Authority* specifies the peer who is responsible for evaluating lit_i or has the authority to evaluate lit_i. The *Authority* argument can be a nested term containing a sequence of authorities, which are then evaluated starting at the outermost layer.

A specific peer may need a way of referring to the peer who asked a particular query. This is accomplished by including a *Requester* argument in literals, so that literals are now of the form

lit_i @ Issuer \$ Requester

The *Requester* argument can also be nested, in which case it expresses a chain of requesters, with the most recent requester in the outermost layer of the nested term.

Using the *Issuer* and *Requester* arguments, it is possible to delegate evaluation of literals to other parties and also express interactions and the corresponding negotiation process between parties (see Fig. 26.3 for an example).

Signed Rules Each peer defines a policy for each of its resources in the form of a set of definite Horn clause rules. These and any other rules that the peer defines on its own are its *local* rules. A peer may also have copies of rules defined by other peers, and it may use these rules to generate proofs, which can be sent to other entities in order to give evidence of the result of a negotiation.

A signed rule has an additional argument that says who signed the rule. The cryptographic signature itself is not included in the policy, because signatures are very large and are not needed by this part of the negotiation software. The signature is used to verify that the issuer really did issue the rule. It is assumed that, when a peer receives a signed rule from another peer, the signature is verified before the rule is passed to the DLP evaluation engine. Similarly, when one peer sends a signed rule to another peer, the actual signed rule must be sent, and not just the logic programmatic representation of the signed rule. More-complex signed rules often represent delegations of authority.

Loop Detection Mechanisms In declarative policy specification, loops may easily occur and should not be considered as errors. For example, declarative policies may state at the same time that "anyone with write permissions can read a file" and "anyone with read permissions can write a file". If not handled accordingly, such loops may end up in nonterminating evaluation [25]. In practice, policies, including for instance business rules, are complex and large in number (and typically not under the control of a single person), which increases the risk of loops and nontermination during dynamic policy evaluation. A distributed tabling algorithm can safely handle mutual recursive dependencies (loops) in distributed environments. Due to the security context, other aspects like private and public policies and proof generation must be taken into account [25].

Protune

The Provisional trust negotiation framework (Protune) [9] aims at combining distributed trust management policies with provisional-style business rules and access-control-related actions. Protune's rule language extends two previous languages: PAPL [22], which until 2002 was one of the most complete policy languages for trust negotiation, and PeerTrust [7], which supports distributed credentials and a more flexible policy protection mechanism. In addition, the framework features a powerful declarative meta-language for driving some critical negotiation decisions, and integrity constraints for monitoring negotiations and credential disclosure.

Protune provides a framework with:

- A trust management language supporting general provisional-style[2] actions (possibly user-defined).
- An extendible declarative meta-language for driving decisions about request formulation, information disclosure, and distributed credential collection.
- A parameterized negotiation procedure, that gives a semantics to the meta-language and provably satisfies some desirable properties for all possible meta-policies.
- Integrity constraints for negotiation monitoring and disclosure control.
- General, ontology-based techniques for importing and exporting meta-policies and for smoothly integrating language extensions.

The Protune rule language is based on normal logic program rules $A \leftarrow L_1, \ldots, L_n$ where A is a standard logical atom (called the *head* of the rule) and $L_1, \ldots, L_n$ (the *body* of the rule) are literals, that is, L_i equals either B_i or $\neg B_i$, for some logical atom B_i.

A *policy* is a set of rules (see Fig. 26.3 for an example), such that negation is applied neither to *provisional predicates* (defined below) nor to any predicate occurring in a rule head. This restriction ensures that policies are *monotonic* on credentials and actions, that is, as more credentials are released and more actions executed, the set of permissions does not decrease.

The vocabulary of predicates occurring in the rules is partitioned into the following categories: *decision predicates* (currently supporting allow() which is queried by the negotiation for access control decisions and sign() which is used to issue statements signed by the principal owning the policy, *abbreviation predicates* (as described in [22]), *constraint predicates* (which

[2] Authorizations involving actions and side effects are sometimes called provisional.

comprise the usual equality and disequality predicates) and *State Predicates* (which perform decisions according to the state). State predicates are further subdivided in *state query predicates* (which read the state without modifying it) and *provisional predicates* (which may be made true by means of associated actions that may modify the current state like, e.g., $credential(), declaration(), logged(X, logfile\_name))$.

Furthermore, meta-policies consist of rules similar to object-level rules. They allow the inspection of terms, check groundness, call an object-level goal G against the current state (using a predicate $holds(G)$), etc. In addition, a set of reserved attributes associated to predicates, literals and rules (e.g., whether a policy is public or sensitive) is used to drive the negotiator's decisions. For example, if p is a predicate, then p.`sensitivity` : `private` means that the extension of the predicate is private and should not be disclosed. An assertion p.`type` : `provisional` declares p to be a provisional predicate; then p can be attached to the corresponding action α by asserting p.`action` :α. If the action is to be executed locally, then we assert p.`actor` : `self`, otherwise we assert p.`actor` : `peer`.

26.2.3 Policy Specification, Conflict Detection and Resolution

Previous sections described how the semantic Web may benefit from the protection of resources with policies specifying security and privacy constraints. However, specifying policies may be as difficult as writing imperative code, getting a policy right is as hard as getting a piece of software correct, and maintaining a large number of them is only harder. Fortunately, the semantic Web can help administrators with policy specification, and detection and resolution of conflicts.

Policy specification Tools like the KAOS policy administration tool (K-PAT) [5] and the PeerTrust policy editor provide an easy-to-use application to help policy writers. This is important because the policies will be enforced automatically and therefore errors in their specification or implementation will allow outsiders to gain inappropriate access to resources, possibly inflicting huge and costly damage. In general, the use of ontologies on policy specification reduces the burden on administrators, helps them with their maintenance, and decreases the number of errors. For example, ontology-based structuring and abstraction help maintain complex software, as they do with complex sets of policies. In the context of the semantic Web, ontologies provide a formal specification of concepts and their interrelationships, and play an essential role in complex Web service environments, semantics-based search engines and digital libraries. Nejdl et al. [14] suggest using two strategies to compose and override policies, building upon the notions of mandatory and default policies, and formalizing the constraints corresponding to these kinds of policies using F-Logic. A prototype implementation as a Protégé plug-in shows that the proposed policy specification mechanism is implementable and effective.

Conflict detection and resolution. semantic Web policy languages also allow for advanced algorithms for conflict detection and its resolution. For example, in Sect. 26.2.1 it was briefly described how conflicts may arise between policies, either at specification time or runtime. A typical example of a conflict is when several policies apply to a request and one allows access while another denies it (positive versus negative authorization). Description logic based languages may use subsumption reasoning to detect conflicts by checking if two policies are instances of conflicting types and whether the action classes that the policies control are not disjoint. Both KAOS and REI handle such conflicts (like right versus prohibition or obligation versus dispensation) within their frameworks and both provide constructs for specifying priorities between policies, hence the most important ones override the less important ones. In addition, REI provides a construct for specifying a general modality priority: positive (rights override prohibitions and obligations override dispensations) or negative (prohibitions override rights and dispensations override obligations). KAOS also provides a conflict resolution technique called policy harmonization. If a conflict is detected the policy with lower priority is modified by refining it with the minimum degree necessary to remove the conflict. This process may generate zero, one or several policies as a refinement of the previous one (see [5] for more information). This process is performed statically at policy specification time, ensuring that no conflicts arise at runtime.

26.3 Applying Policies on the Semantic Web

The benefits of using semantic policy languages in distributed environments with automated machine–machine interaction have been described extensively in previous sections. This section aims at providing some examples of its use in the context of the Web, (semantic) Web Services and the (semantic) grid. In all cases, different solutions have been described addressing different scenarios from the point of view of one-step authorization or policy-driven negotiations.

26.3.1 Policies on the Web

The current Web infrastructure does not allow the enforcement of user policies while accessing Web resources. Web server authentication is typically based on authentication mechanisms in which users must authenticate themselves (either by means of certificates or typing a user name and password). Semantic Web policies overcome such limitations of the Web.

Kagal et al. [6] describe how the REI language can be applied in order to control access to Web resources. Web pages are marked up with policies specifying which credentials are required to access such pages. A policy engine (bound to the webserver) decides whether the request matches the credentials requested. In case it does not, the webserver could show which credentials are missing. Furthermore, Kolari et al. [26] presents an extension to the platform for privacy preferences (P3P) using the REI language. The authors propose enhancements using REI policies to increase expressiveness and to allow for existing privacy enforcement mechanisms.

PeerTrust can be used to provide advanced policy-driven negotiations on the Web in order to control access to resources [7, 27]. A user receives a signed (by a trusted authority) applet after requesting access to a resource. Such an applet includes reasoning capabilities and is loaded in the Web browser. The applet automatically imports the policies specified by the user and starts a negotiation. If the negotiation succeeds, the applet simply retrieves the resource requested or, if necessary, redirects the user to the appropriate repository.

26.3.2 Semantic Web Services

Semantic Web services aim at the automation of discovery, selection and composition of Web services. Denker et al. [28] and Kagal et al. [11] suggest extending OWL-S with security policies, written in REI, like e.g., whether a service requires or is capable of providing secure communication channels. An agent may then submit a request to the registry together with its privacy policies. The matchmaker at the registry will filter out incompatible service descriptions and select only those whose security requirements of the service match the privacy policies of the requester.

Differently, Olmedilla et al. [29] propose the use of the PeerTrust language to decide if trust can be established between a requester and a service provider during runtime selection of Web services. Modelling elements are added to the Web service modelling ontology (WSMO) in order to include security information in the description of Semantic Web Services. In addition, the authors discuss different registry architectures and their implications for the matchmaking process.

26.3.3 Semantic Grid

Grid environments provide the middleware needed to access distributed computing and data resources. Distinctly administrated domains form virtual organizations and share resources for data retrieval, job execution, monitoring, and data storage. Such an environment provides users with seamless access to all resources they are authorized to access. In current Grid infrastructures, in order to be granted access at each domain, user's jobs have to secure and provide appropriate digital credentials for authentication and authorization. However, while authentication along with single sign-on can be provided based on client delegation of X.509 proxy certificates to the job being submitted, the authorization mechanisms are still mainly identity-based. Due to the large number of potential users and different certification authorities, this leads to scalability problems calling for a complementary solution to the access control mechanisms specified in the current grid security infrastructure (GSI) [30].

Uszok et al. [31] present an integration of the KAOS framework into Globus Tookit 3. Its authors suggest offering a KAOS grid service and providing an interface so grid clients and services may register and check whether a specific action is authorized or not. The KAOS grid service uses the KAOS policy services described in Sect. 26.2.1 and relies on the Globus local enforcement mechanisms.

Alternatively, Constandache et al. [32] describe an integration of policy-driven negotiations for the GSI, using semantic policies and enhancing it by providing automatic credential fetching and disclosure. Policy-based dynamic negotiations allow more-flexible authorization in complex Grid environments, and relieve both users and administrators from up-front negotiations and registrations. Constandache et al. [32] introduce an extension to the GSI and Globus Toolkit 4.0 in which policy-based negotiation mechanisms offer the basis for overcoming these limitations. This extension includes property-based authorization mechanisms, automatic gathering of required certificates, bidirectional and iterative trust negotiation and policy-based authorization, ingredients that provide advanced self-explanatory access control to grid resources.

26.4 Open Research Issues

Although there has been extensive research in recent years, there exist still open issues that must be solved [33]. The following provides a nonexhaustive list of issues which have not yet been given enough attention, or that still remain unsolved and crucial challenges in order to have a semantic policy framework adopted in real-world applications.

- Adoption of a *broad notion of policy*, encompassing not only access control policies, but also privacy policies, business rules, quality of service, agent conversation, mobility policies, etc. All these different kinds of policies should eventually be integrated into a single framework.
- *Strong and lightweight evidence*: Policies make decisions based on the properties of the peers interacting with the system. These properties may be strongly certified by cryptographic techniques, or may be reliable to some intermediate degree with lightweight evidence gathering and validation. A flexible policy framework should try to merge these two forms of evidence to meet the efficiency and usability requirements of Web applications. Independently to prevention techniques, audits can be explored to detect malicious behaviour (see Chaps. 24 and 25 for more details).
- These desiderata imply that trust negotiation, reputation models, business rules, and action specification languages have to be integrated into a single framework at least to some extent. It is crucial to find the right tradeoff between generality and efficiency.
- *Automated policy-driven negotiation* is one of the main ingredients that can be used to make heterogeneous peers effectively interoperate.

- *Lightweight knowledge representation and reasoning* does not only refer to computational complexity; it should also reduce the effort to specialize general frameworks to specific application domains; and the corresponding tools should be easy to learn and use for common users, with no particular training in computers or logic.
- The last issue cannot be tackled simply by adopting a rule language. Solutions like *controlled natural-language syntax for policy rules*, to be translated by a parser into the internal logical format, will definitively ease the adoption of any policy language.
- *Cooperative policy enforcement*: A secure cooperative system should (almost) never say *no*. Web applications need to help new users in obtaining the services that the application provides, so potential customers should not be discouraged. Whenever prerequisites for accessing a service are not met, Web applications should explain what is missing and help the user to obtain the required permissions. As part of cooperative enforcement, advanced *explanation mechanisms* are necessary to help users understand policy decisions and obtaining the permission to access a desired service.

26.5 Conclusions

This chapter provides an introduction to policy-based security and privacy management on the semantic Web. It describes the benefits of using policies and presents four of the most relevant policy languages in the semantic Web context. These four languages are classified according to whether policies are assumed to be public or else may be protected. The former consists of a single evaluation step where a policy engine or a matchmaker decides whether two policies are compatible or not. Examples of this kind of evaluation are the KAOS and REI frameworks. If policies may be protected (by e.g., other policies), the process is no longer a one-step evaluation. In this case, policies guide a negotiation in which policies are disclosed iteratively increasing the level of security at each step towards a final agreement. Examples of these kind of frameworks are PeerTrust and Protune. Furthermore, semantic Web techniques can be used to ease and enhance the process of policy specification and validation. Conflicts between policies can be found and even resolved automatically (either by meta-policies or by harmonization algorithms).

In order to demonstrate the benefits and feasibility of semantic Web policies, several application scenarios are described, namely the Web, (semantic) Web Services and the (semantic) grid. Finally the chapter concludes with a list of open research issues that prevent existing policy languages from being widely adopted. This list is intended to help new researchers in the area to focus on those crucial problems which are still unsolved.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.
2. G. Antoniou, M. Baldoni, P.A. Bonatti, W. Nejdl, and D. Olmedilla. Rule-based policy specification. In Ting Yu and Sushil Jajodia, editors, *Decentralized Data Management Security*. Springer, 2006.
3. M. Blaze, J. Feigenbaum, and A.D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In *Security Protocols, 6th International Workshop*, volume 1550 of *LNCS*, pages 59–63, Cambridge, April, 1998. Springer.
4. M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the policy-maker trust management system. In *Financial Cryptography, Second International Conference*, volume 1465 of *LNCS*, pages 254–274, Anguilla, British West Indies, February 1998. Springer.
5. A. Uszok, J.M. Bradshaw, R. Jeffers, N. Suri, P.J. Hayes, M.R. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *POLICY*, page 93, 2003.
6. L. Kagal, T.W. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, LNCS, pages 402–418. Springer, 2003.
7. R. Gavriloaie, W. Nejdl, D. Olmedilla, K.E. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *1st European Semantic Web Symposium (ESWS 2004)*, volume 3053 of *LNCS*, pages 342–356, Heraklion, Crete, Greece, May 2004. Springer.
8. M.Y. Becker and P. Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004), 7-9 June 2004, Yorktown Heights, NY, USA*, pages 159–168. IEEE Computer Society, 2004.
9. P. A. Bonatti and D. Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 14–23, Stockholm, Sweden, 2005. IEEE Computer Society.
10. G. Tonti, J.M. Bradshaw, R. Jeffers, R.Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In *International Semantic Web Conference*, pages 419–437, 2003.
11. L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. W. Finin, and K.P. Sycara. Authorization and privacy for semantic web services. *IEEE Intelligent Systems*, 19(4):50–56, 2004.
12. K. Taveter and G. Wagner. Agent-oriented enterprise modeling based on business rules. In *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling*, pages 527–540. Springer-Verlag, 2001.
13. W.H. Winsborough, K.E. Seamons, and V.E. Jones. Automated trust negotiation. DARPA Information Survivability Conference and Exposition, IEEE Press, Jan 2000.

14. W. Nejdl, D. Olmedilla, M. Winslett, and C.C. Zhang. Ontology-based policy specification and management. In *2nd European Semantic Web Conference (ESWC)*, volume 3532 of *LNCS*, pages 290–302, Heraklion, Crete, Greece, May 2005. Springer.
15. M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, LNCS, pages 351–368. Springer, 2003.
16. J. Golbeck and J.A. Hendler. Accuracy of metrics for inferring trust and reputation in semantic web-based social networks. In *Engineering Knowledge in the Age of the Semantic Web, 14th International Conference, EKAW 2004, Whittlebury Hall, UK, October 5-8, 2004, Proceedings*, LNCS, pages 116–131. Springer, 2004.
17. J. Golbeck, B. Parsia, and J.A. Hendler. Trust networks on the semantic web. In *Cooperative Information Agents VII, 7th International Workshop, CIA 2003, Helsinki, Finland, August 27-29, 2003, Proceedings*, LNCS, pages 238–249. Springer, 2003.
18. J.M. Bradshaw, A. Uszok, R. Jeffers, N. Suri, P. J. Hayes, M.H. Burstein, A. Acquisti, B. Benyo, M. R. Breedy, M.M. Carvalho, D.J. Diller, M. Johnson, S. Kulkarni, J. Lott, M. Sierhuis, and R. van Hoof. Representation and reasoning for DAML-based policy and domain services in KAoS and nomads. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, Melbourne, Victoria, Australia, July 2003.
19. M. Dean and G. Schreiber. OWL web ontology language reference, 2004.
20. F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
21. L. Kagal. *A Policy-Based Approach to Governing Autonomous Behaviour in Distributed Environments.* PhD thesis, University of Maryland Baltimore County, 2004.
22. P. Bonatti and P. Samarati. Regulating Service Access and Information Release on the Web. In *Conference on Computer and Communications Security (CCS'00)*, Athens, November 2000.
23. N. Li and J.C. Mitchell. RT: A Role-based Trust-management Framework. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, Washington, D.C., April 2003.
24. J. Trevor and D. Suciu. Dynamically distributed query evaluation. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Santa Barbara, CA, USA, May 2001.
25. M. Alves, C. Viegas Damásio, D. Olmedilla, and W. Nejdl. A distributed tabling algorithm for rule based policy systems. In *7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2006)*, London, Ontario, Canada, 2006. IEEE Computer Society.
26. P. Kolari, L. Ding, S. Ganjugunte, A. Joshi, T.W. Finin, and L. Kagal. Enhancing web privacy protection through declarative policies. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005)*, pages 57–66, Stockholm, Sweden, June 2005. IEEE Computer Society.
27. S. Staab, B.K. Bhargava, L. Lilien, A. Rosenthal, M. Winslett, M. Sloman, T.S. Dillon, E. Chang, F.K. Hussain, W. Nejdl, D. Olmedilla, and V. Kashyap. The pudding of trust. *IEEE Intelligent Systems*, 19(5):74–88, 2004.

28. G. Denker, L. Kagal, T. W. Finin, M. Paolucci, and K.P. Sycara. Security for daml web services: Annotation and matchmaking. In *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, LNCS, pages 335–350. Springer, 2003.
29. D. Olmedilla, R. Lara, A. Polleres, and H. Lausen. Trust negotiation for semantic web services. In *1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, volume 3387 of *LNCS*, pages 81–95, San Diego, CA, USA, July 2004. Springer.
30. Grid Security Infrastructure. http://www.globus.org/security/overview.html.
31. A. Uszok, J.M. Bradshaw, and R. Jeffers. Kaos: A policy and domain services framework for grid computing and semantic web services. In *Trust Management, Second International Conference, iTrust 2004, Oxford, UK, March 29 - April 1, 2004, Proceedings*, LNCS, pages 16–26. Springer, 2004.
32. I. Constandache, D. Olmedilla, and W. Nejdl. Policy based dynamic negotiation for grid services authorization. In *Semantic Web Policy Workshop in conjunction with 4th International Semantic Web Conference*, Galway, Ireland, November 2005.
33. P.A. Bonatti, C. Duma, N. Fuchs, W. Nejdl, D. Olmedilla, J. Peer, and N. Shahmehri. Semantic Web policies - A discussion of requirements and research issues. In *3rd European Semantic Web Conference (ESWC)*, Lecture Notes in Computer Science, Budva, Montenegro, June 2006.