Rule-Based Policy Representation and Reasoning for the Semantic Web

Piero A. Bonatti and Daniel Olmedilla

[1] Università di Napoli Federico II, Napoli, Italy
bonatti@na.infn.it
[2] L3S Research Center and University of Hannover
olmedilla@L3S.de

Summary. The Semantic Web aims at enabling 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 an automatic access to sensitive information. Policy-based access control provides sophisticated means in order to support protecting sensitive resources and information disclosure.

However, the term policy is often overloaded. A general definition might be "a statement that defines the behaviour of a system". However, such a general definition encompasses different notions, including security policies, trust management policies, business rules and quality of service specifications, just to name a few. Researchers have mainly focussed on one or more of such notions separately but not on a comprehensive view. Policies are pervasive in web applications and play crucial roles in enhancing security, privacy, and service usability as well. Interoperability and self-describing semantics become key requirements and here is where Semantic Web comes into play. There has been extensive research on policies, also in the Semantic Web community, but there still exist some issues that prevent policy frameworks from being widely adopted by users and real world applications.

This document aims at providing an overall view of the state of the art (requirements for a policy framework, some existing policy frameworks/languages, policy negotiation, context awareness, etc.) as well as open research issues in the area (policy understanding in a broad sense, integration of trust management, increase in system cooperation, user awareness, etc.) required to develop a successful Semantic Policy Framework.

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 right 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 search). 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 infomation. 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 the website states the conditions to be satisfied in order to reserve a room but also the user agent may 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., 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 authorisation 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 completely solved them yet. Examples like KeyNote [3] or PolicyMaker [4] 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. RBAC (Role-Based Access Control) also does not meet Semantic Web requirements since it is difficult to assign roles to users which are not known in advance. Regarding to user's privacy protection, Platform for Privacy Preferences (P3P) provides a standard vocabulary to describe Web server policies. However, it is not expressive enough (it is a schema, not a language, and only describes purpose for the gathered data) and it does not allow for enforcement mechanisms. On the other hand, there is a wide offer of policy languages that have been developed to date [5, 6, 7, 8], addressing the general requirements for a Semantic Web policy language: expressiveness, simplicity, enforceability, scalability, and analyzability [9]. 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, 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 in gathering the relevant policies (and possibly relevant credentials) from the involved entities and checking whether they *match* (e.g., [10]). On the other hand, if policies may be private (typical situation for business rules [11]), 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 [12].

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 on specification, conflict detection and resolution of such policies [5, 13].

As it can be seen, there has been extensive research in the area, including the Semantic Web community, but several aspects still exist that prevent policy frameworks from widespread adoption and real world application. This manuscript incorporates and merges the ideas of previosly published papers [14, 2, 15] and aims at providing an overall view of the state of the art (requirements for a policy framework, some existing policy frameworks/languages, policy negotiation, context awareness, etc.) as well as open research issues in the area (policy understanding in a broad sense, integration of trust management, increase in system cooperation, user awareness, etc.) required to develop a successful Semantic Policy Framework. Section 2 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. Some examples of application scenarios are presented in Section 3, where policy based security and privacy are used. Section 4 discusses important requirements and open research issues in this context, focusing on policies in general and their integration into trust management frameworks, as well as on approaches to increase system cooperation, usability and user-awareness of policy issues. This manuscript finally concludes with a last section (Section 5) in which the most important isssues presented are summarized.

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 web server 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 (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 negotiation in which new policies and credentials are disclosed at each step, therefore 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 1. Additionally, we use the running policy "only employees of institution XYZ may retrieve a file" to illustrate an example of each language.

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 in case 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 1. Comparison of KAOS, REI, PeerTrust and Protune[3]

Policy Language	Authorization Protocol	Reasoning Paradigm	Conflict Detection	Meta-policies	Loop Detection
KAOS	One-step	DL	Static detection & resolution		
REI	One-step	DL + variables	Dinamyc detection & 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, 16] provide a framework for specification, management, conflict resolution and enforcement of policies allowing for distributed policy interaction and support for dynamic policy changes. It uses OWL [17] 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) authorisation,

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 (Figure 1 sketches part of a KAOS policy).

KAOS benefits from the OWL representation and description logic based subsumption mechanisms [18]. Thus, it allows 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 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 authorisation (a policy allows access and but another denies it), positive vs. negative obligation (a policy obliges to execute an action while another dispensates from such obligation) and positive obligation vs. negative authorisation (a policy obliges to execute an action but another denies authorisation for such execution). KAOS resolves such conflics (also called harmonisation) based on assigning preferences to policies and resolving in favour of the policies with higher priority (Section 2.3 will later extend on this).

Finally, KAOS assumes a default authorisation 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 authorised".

```
<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>
```

Fig. 1. Example of KAOS policies

[3] DL refers to Description Logic while LP stands for Logic Programming

REI

REI 2.0 [19, 10] 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 Figure 2 for an example) are described in terms of deontic concepts: permissions, prohibitions, obligations and dispensations, equivalently to the positive/negative authorisations 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 vs. prohibition or obligation vs. dispensation). REI provides mechanisms for conflict detection and constructs to resolve them, namely, overriding policies (similar to the prioritisation in KAOS) and definition at the meta-level of the global modality (positive or negative) that holds (see Section 2.3 for more details).

```
<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. 2. Example of REI policies

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 authorisation from Alice's psychiatrist. Another example, imagine Tom wants to share his holiday pictures on-line

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 a several steps negotiation, by disclosing new policies and credentials at each step, and therefore advancing after each iteration towards a common agreement [12]. 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 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 Figure 3).

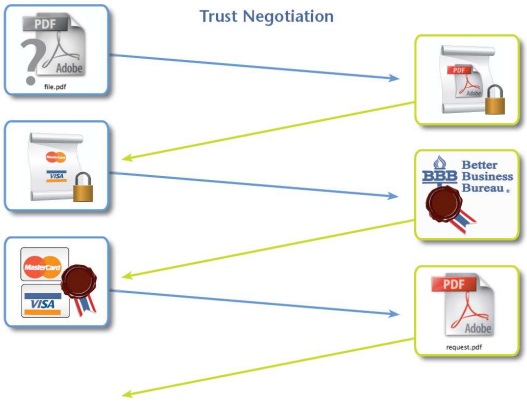

Fig. 3. Policy-driven negotiation between Alice and e-shop

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 [20, 21, 22], which may be applied to the Semantic Web.

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 are only described 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.
```

Fig. 4. Example of PeerTrust 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 now literals are 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 Figure 4 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 non-terminating evaluation [23]. In practice, policies, including for instance business rules, are complex and large in number (and typically not under control of a single person) which increases the risk of loops and non-termination during dynamic policy evaluation. A distributed tabling algorithm can handle safely 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 [23].

Protune

The PRovisional TrUst NEgotiation framework Protune [8] 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 [20], 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.

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

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

Fig. 5. Example of Protune policies

Protune provides a framework with:

- A trust management language supporting general provisional-style[4] actions (possibly user-defined).

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

- An extendible declarative meta-language for driving decisions about request formulation, information disclosure, and distributed credential collection.
- A parameterised 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.
- Advanced policy explanations in order to answer why, why-not, how-to, and what-if queries [24]

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 Figure 5 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 [20]), *Constraint Predicates* (which 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 to inspect 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 assert p.`actor` : `peer`.

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 damages. 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, and so do they 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. [13] suggest using two strategies to compose and override policies, building upon the notions of mandatory and default policies, and formalising 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 Section 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 vs. negative authorisation). 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 vs. prohibition or obligation vs. 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 harmonisation'. 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.

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 extensible 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.

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 web server) decides whether the request matches the credentials requested. In case it does not, the web server could show which credentials are missing. Furthermore, Kolari et al. [25] presents an extension to the Platform for Privacy Preferences (P3P) using the REI language. The authors propose enhancements using REI policies to increase the 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, 26]. 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 retrieve the resource requested or, if necessary, redirects the user to the appropriate repository.

3.2 Semantic Web Services

Semantic Web Services aim at the automation of discovery, selection and composition of Web Services. Denker et al. [27] and Kagal et al. [10] 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 non-compatible service descriptions and select only those whose security requirements of the service match the privacy policies of the requester.

Differently, Olmedilla et al. [28] 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 Modeling 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.

3.3 Semantic Grid

Grid environments provide the middleware needed for access distributed computing and data resources. Distinctly administrated domains form virtual organisations 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 authorised 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 authorisation. 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 authorisation 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) [29].

Uszok et al. [30] presents 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 authorised or not. The KAOS grid service uses the KAOS policy services described in Section 2.1 and relies on the Globus local enforcement mechanisms.

Alternatively, Constandache et al. [31] describe an integration of policy driven negotiations for the GSI, using semantic policies and enhancing it providing automatic credential fetching and disclosure. Policy-based dynamic negotiations allow more flexible authorisation in complex Grid environments, and relieve both users and administrators from up front negotiations and registrations. Constandache et al. [31] introduces 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 authorisation mechanisms, automatic gathering of required certificates, bidirectional and iterative trust negotiation and policy based authorisation, ingredients that provide advanced self-explanatory access control to grid resources.

4 Requirements and Open Research Issues for a Semantic Web Policy Framework

Policies are pervasive in web applications. They play crucial roles in enhancing security, privacy and usability of distributed services, and indeed may determine the success (or failure) of a web service. However, users will not be able to benefit from these protection mechanisms unless they understand and are able to personalize policies applied in such contexts. For web services this includes policies for access control, privacy and business rules, among others.

This section summarizes research performed over the past years on semantic policies and especially aim to analyse those aspects that did not receive so much attention so far. We will focus our discussion on the following strategic goals and lines of research:

- *Rules-based policy representation*: Rule-based languages are commonly regarded as the best approach to formalizing policies due to its flexibility, formal semantics and closeness to the way people think.
- Adoption of a *broad notion of policy*, encompassing not only access control policies, but also privacy policies, business rules, quality of service, and others. We believe that all these different kinds of policies should eventually be integrated into a single framework.
- *Strong and lightweight evidence*: Policies make decisions based on 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.
- 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. *So far, no framework has tried to merge all aspects into a coherent system.*
- *Automated trust negotiation* is one of the main ingredients that can be used to make heterogeneous peers effectively interoperate. This approach relies on and actively contributes to advances in the area of *trust management*.
- *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. We regard these properties as crucial for the success of a semantic web framework.
- 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 in obtaining the required permissions.
- As part of cooperative enforcement, advanced *explanation mechanisms* are necessary to help users in understanding policy decisions and obtaining the permission to access a desired service.

In the remainder of this section we describe the current state of the art on these issues, expand on them and point out several interesting research directions related to them: the need for an flexible and easy policy representation, the different types of policies which must be considered in order to address real world scenarios, the need for strong and lightweight evidence on the information that policies require, the importance of trust management as part of a policy framework, describing in detail negotiations and provisional actions and how cooperative systems which explain their decisions to users as well as policy specification in natural language increase user awareness and understanding.

4.1 Rule-based policy representation

Rule-based languages are commonly regarded as the best approach to formalizing security policies. In fact, most of the systems we use every day adopt policies formulated as rules. Roughly speaking, the access control lists applied by routers are actually rules of the form: "*if a packet of protocol X goes from hosts Y to hosts Z then [don't] let it pass*". Some systems, like Java, adopt procedural approaches. Access control is enforced by pieces of code scattered around the virtual machine and the application code; still, the designers of Java security felt the need for a method

called *implies*, reminiscent of rules, that causes certain authorizations to entail other authorizations [32].

The main advantages of rule-based policy languages can be summarized as follows:

- People (including users with no specific training in computers or logic) spontaneously tend to formulate security policies as rules.
- Rules have precise and relatively simple formal semantics, be it operational (rewrite semantics), denotational (fixpoint-based), or declarative (model theoretic). Formal semantics is an excellent help in implementing and verifying access control mechanisms, as well as validating policies.
- Rule languages can be flexible enough to model in a unified framework the many different policies introduced along the years as ad-hoc mechanisms. Different policies can be harmonized and integrated into a single coherent specification.

In particular, logic programming languages are particularly attractive as policy specification languages. They enjoy the above properties and have efficient inference mechanisms (linear or quadratic time). This property is important as in most systems policies have to manage a large number of users, files, and operations—hence a large number of possible authorizations. And for those applications where linear time is too slow, there exist well-established compilation techniques (materialization, partial evaluation) that may reduce reasoning to pure retrieval at run time.

Another fundamental property of logic programs is that their inference is *nonmonotonic*, due to *negation-as-failure*. Logic programs can make default decisions in the absence of complete specifications. Default decisions arise naturally in real-world security policies. For example, *open* policies prescribe that authorizations by default are granted, whereas *closed* policies prescribe that they should be denied unless stated otherwise. Other nonmonotonic inferences, such as authorization inheritance and overriding, are commonly supported by policy languages.

For all of these reasons, rule languages based on nonmonotonic logics eventually became the most frequent choice in the literature. A popular choice consists of *normal logic programs*, i.e. sets of rules like

$$A \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n$$

interpreted with the *stable model semantics* [33]. In general, each program may have one stable model, many stable models, or none at all. There are opposite points of view on this feature.

Some authors regard multiple models as an opportunity to write nondeterministic specifications where each model is an acceptable policy and the system makes an automatic choice between the available alternatives [34]. For instance, the models of a policy may correspond to all possible ways of assigning permissions that preserve a *Chinese Wall* policy [35]. However, the set of alternative models may grow exponentially, and the problem of finding one of them is NP-complete. There are exceptions with polynomial complexity [36, 37], though.

Some authors believe that security managers would not trust the system's automatic choice and adopt restrictions such as *stratifiability* [38] to guarantee that the canonical model be unique. The system rejects non-stratified specifications, highlighting nonstratified rules to help the security administrator in reformulating the specifications. As a further advantage, stratifiability-like restrictions yield PTIME semantics.

4.2 A broad notion of policy

Policies are pervasive in all web-related contexts. Access control policies are needed to protect any system open to the internet. Privacy policies are needed to assist users while they are browsing the web and interacting with web services. Business rules specify which conditions apply to each customer of a web service. Other policies specify constraints related to Quality of Service (QoS). In E-government applications, visas and other documents are released according to specific eligibility policies. This list is not exhaustive and is limited only by the class of applications that can be deployed in the world wide web.

Most of these policies make their decisions based on similar pieces of information [39] – essentially, properties of the peers involved in the transaction. For example, age, nationality, customer profile, identity, and reputation may all be considered both in access control decisions, and in determining which discounts are applicable (as well as other eligibility criteria). It is appealing to integrate these kinds of policies into a coherent framework, so that (i) a common infrastructure can be used to support interoperability and decision making, and (ii) the policies themselves can be harmonized and synchronized.

In the general view depicted above, policies may also establish that some events must be logged (audit policies), that user profiles must be updated, and that when a transaction fails, the user should be told how to obtain missing permissions. In other words, policies may specify *actions* whose execution may be interleaved with the decision process. Such policies are called *provisional policies*. In this context, *policies act both as decision support systems and as declarative behavior specifications*. An effective user-friendly approach to policy specification could give common users (with no training in computer science or logic) better control on the behavior of their own system (see the discussion in Section 4.5).

Of course, the extent to which this goal can be achieved depends on the policy's ability to *interoperate* with legacy software and data – or more generally, with the rest of the system. Then a policy specification language should support suitable primitives for interacting with external packages and data in a flexible way.

The main challenges raised by these issues are then the following:

- Harmonizing security and privacy policies with business rules, provisional policies, and other kinds of policy is difficult because their standard formalizations are based on different derivation strategies, and even different reasoning mechanisms (cf. Section 4.4). Deduction, abduction, and event-condition-action rule semantics need to be integrated into a coherent framework, trying to minimize subtleties and technical intricacies (otherwise the framework would not be accessible to common users).
- Interactions between a rule-based theory and "external" software and data have been extensively investigated in the framework of logic-based mediation and logic-based agent programming [40, 41]. However, there are novel issues related to implementing high-level policy rules with low-level mechanisms such as firewalls, web server and DBMS security mechanisms, and operating system features, that are often faster and more difficult to bypass than rule interpreters [42]. A convincing realization of this approach might boost the application of the rich and flexible languages developed by the security community.

4.3 Strong and lightweight evidence

Currently two major approaches for managing trust exist: policy-based and reputation-based trust management. The two approaches have been developed within the context of different environments and target different requirements. On the one hand, policy-based trust relies on "strong security" mechanisms such as signed certificates and trusted certification authorities (CAs) in order to regulate access of users to services. Moreover, access decisions are usually based on mechanisms with well defined semantics (e.g., logic programming) providing strong verification and analysis support. The result of such a policy-based trust management approach usually consists of a binary decision according to which the requester is trusted or not, and thus the service (or resource) is allowed or denied. On the other hand, reputation-based trust relies on a "soft computational" approach to the problem of trust. In this case, trust is typically computed from local experiences together with the feedback given by other entities in the network. For instance, eBay buyers and sellers rate each other after each transaction. The ratings pertaining to a certain seller (or buyer) are aggregated by eBay's reputation system into a number reflecting seller (or buyer) trustworthiness as judged by the eBay community. The reputation-based approach has been favored for environments such as Peer-to-Peer or Semantic Web, where the existence of certifying authorities can not always be assumed but where a large pool of individual user ratings is often available.

Another approach – very common in today's applications – is based on forcing users to commit to contracts or copyrights by having users click an "accept" button on a pop-up window. This is perhaps the lightest approach to trust, that can be generalized by having users utter *declarations* (on their e-mail address, on their preferences, etc.) e.g. by filling an HTML form.

Real life scenarios often require to make decisions based on a combination of these approaches. Transaction policies must handle expenses of all magnitudes, from micropayments (e.g. a few cents for a song downloaded to your iPod) to credit card payments of a thousand euros (e.g. for a plane ticket) or even more. The cost of the traded goods or services contributes to determine the risk associated to the transaction and hence the trust measure required.

Strong evidence is generally harder to gather and verify than lightweight evidence. Sometimes, a "soft" reputation measure or a declaration in the sense outlined above is all one can obtain in a given scenario. We believe that the success of a trust management framework will be determined by the ability of *balancing trust levels and risk levels* for each particular task supported by the application, adding the following to the list of interesting research directions:

- How should different forms of trust be integrated? Some hints on modelling context aware trust, recommendation and risk with rules is given in [26] and a first proposal for a full integration in a policy framework can be found in [43]. However, new reputation models are being introduced, and there is a large number of open research issues in the reputation area (e.g., vulnerability to coalitions). Today, it is not clear which of the current approaches will be successful and how the open problems will be solved. Any proposal should therefore aim at maximal modularity in the integration of numerical and logical trust.
- How many different forms of evidence can be conceived? In principle, properties of (and statements about) an individual can be extracted from any – possibly unstructured – web resource. Supporting such a variety of information in policy

decisions is a typical semantic web issue – and an intriguing one. However, such general policies are not even vaguely as close to become real as the policies based on more "traditional" forms of evidence (see the discussion in the next section).

4.4 Trust management

During the past few years, some of the most innovative ideas on security policies arose in the area of *automated trust negotiation* [44, 8, 45, 7, 46, 47, 48, 49, 50]. That branch of research considers peers that are able to automatically negotiate credentials according to their own declarative, rule-based policies. Rules specify for each resource or credential request which properties should be satisfied by the subjects and objects involved. At each negotiation step, the next credential request is formulated essentially by *reasoning* with the policy, e.g. by inferring implications or computing abductions.

Since about five years frameworks exist where credential requests are formulated by exchanging *sets of rules* [8, 45]. Requests are formulated *intensionally* in order to express compactly and simultaneously all the possible ways in which a resource can be accessed — shortening negotiations and improving privacy protection because peers can choose the best option from the point of view of sensitivity. It is not appealing to request "*an ID and a credit card*" by enumerating all possible pairs of ID credentials and credit card credentials; it is much better to *define* what IDs and credit cards are and send the definition itself. Another peer may use it to check whether some subset of its own credentials fulfills the request. This boils down to gathering the relevant concept definitions in the policy (so-called *abbreviation rules*) and sending them to the other peer that reasons with those rules locally.

In [8, 45] *peers communicate by sharing their ontologies*. Interestingly, typical policies require peers to have a common a priori understanding only of the predicate representing credentials and arithmetic predicates, as any other predicate can be understood by sharing its definition. The only nontrivial knowledge to be shared is the X.509 standard credential format. In this framework, interoperability based on ontology sharing is already at reach! This is one of the aspects that make policies and automated trust negotiation a most attractive application for semantic web ideas.

Another interesting proposal of [45] is the notion of *declaration*, that has already been discussed in Section 4.3. This was the first step towards a more flexible and lightweight approach to policy enforcement, aiming at a better tradeoff between protection efforts and risks. According to [51], this framework was one of the most complete trust negotiation systems. The major limitation was the lack of distributed negotiations and credential discovery, which are now supported as specified in [8].

Negotiations

In response to a resource request, a web server may ask for credentials proving that the client can access the resource. However, the credentials themselves can be sensitive resources. So the two peers are in a completely symmetrical situation: the client, in turn, asks the server for credentials (e.g. proving that it participates in the Better Business Bureau program) before sending off the required credentials. Each peer decides how to react to incoming requests according to a local policy, which is typically a set of rules written in some logic programming dialect. As we

pointed out, requests are formulated by selecting some rules from the policies. This basic schema has been refined along the years taking several factors into account [44, 8, 45, 7, 46, 47, 48, 49, 50].

First, policy rules may possibly inspect a *local state* (such as a legacy database) that typically is not accessible by other peers. In that case, in order to make rules intelligible to the recepient, they are partially evaluated with respect to the current state.

Second, *policies themselves are sensitive resources*, therefore not all relevant rules are shown immediately to the peer. They are first filtered according to policy release rules; the same schema may be applied to policy release rules themselves for an arbitrary but finite number of levels. As a consequence, some negotiations that might succeed, in fact fail just because the peers do not tell each other what they want. The study of methodologies and properties that guarantee negotiation success is an interesting open research issue.

Moreover, *credentials are not necessarily on the peer's host*. It may be necessary to locate them on the network [52]. As part of the automated support to *cooperative enforcement*, peers may give each other hints on where a credential can be found [53].

There are further complications related to actions (cf. Section 4.4). In order to tune the negotiation strategy to handle these aspects optimally, we can rely on a *metapolicy language* [8] that specifies which predicates are sensitive, which are associated to actions, which peer is responsible for each action, and where credentials can be searched for, guiding negotiation in a declarative fashion and making it more cooperative and interoperable. Moreover, the metapolicy language can be used to instantiate the framework in different application domains and link predicates to the ontologies where they are defined.

Provisional policies

Policies may state that certain requests or decisions have to be logged, or that the system itself should search for certain credentials. In other words, policy languages should be able to specify *actions*. Event-condition-action (ECA) rules constitute one possible approach. Another approach consists in labelling some predicates as *provisional*, and associating them to actions that (if successful) make the predicate true [8]. We may also specify that an action should be executed by some other peer; this results in a request.

A cooperative peer tries to execute actions under its responsibility whenever this helps in making negotiations succeed. For example, provisional predicates may be used to encode business rules. The next rule[5] enables discounts on low_selling articles in a specific session:

$$\begin{aligned}&\texttt{allow}(Srv) \leftarrow \ldots, \texttt{session}(ID),\\ &\quad \texttt{in}(X, \texttt{sql:query}(\texttt{'select} * \texttt{from low\_selling'})),\\ &\quad \texttt{enabled}(\texttt{discount}(X), ID)\,.\end{aligned}$$

Intuitively, if $\texttt{enabled}(\texttt{discount}(X), ID)$ is not yet true but the other conditions are verified, then the negotiator may execute the action associated to `enabled` and

[5] formulated in PROTUNE's language

the rule becomes applicable (if `enabled(discount(`X`),` ID`)` is already true, no action is executed). The (application dependent) action can be defined and associated to `enabled` through the metapolicy language. With the metalanguage one can also specify when an action is to be executed.

Some actions would be more naturally expressed as ECA rules. However, it is not obvious how the natural bottom-up evaluation schema of ECA rules should be integrated with the top-down evaluation adopted by the current core policy language. The latter fits more naturally the abductive nature of negotiation steps. So integration of ECA rules is still an interesting open research issue.

Stateful vs. stateless negotiations

Negotiations as described above are in general stateful, because (i) they may refer to a local state – including legacy software and data – and (ii) the sequence of requests and counter requests may become more efficient if credentials and declarations are not submitted again and again, but kept in a local negotiation state. However, negotiations are not *necessarily* stateful because

- the server may refuse to answer counter-requests, or – alternatively – the credentials and declarations disclosed during the transaction may be included in every message and need not be cached locally;
- the policy does not necessarily refer to external packages.

Stateless protocols are just special cases of the frameworks introduced so far. Whether a stateless protocol is really more efficient depends on the application. Moreover, efficiency at all costs might imply less cooperative systems.

Are stateful protocols related to scalability issues? We do not think so. The web started as a stateless protocol, but soon a number of techniques were implemented to simulate stateful protocols and transactions in quite a few real world applications and systems, capable of answering a huge number of requests per time unit. We observe that if the support for stateful negotiations had been cast into http, probably many of the intrinsic vulnerabilities of simulated solutions (like cookies) might have been avoided.

New Issues

Existing approaches to trust management and trust negotiation already tackle the need for flexible, knowledge-based interoperability, and take into account the main idiosyncrasies of the web – because automated trust negotiation frameworks have been designed with exactly that scenario in mind. Today, to make a real contribution (even in the context of a policy-aware web), we should further perform research on the open issues of trust management, including at least the following topics:

- Negotiation success: how can we guarantee that negotiations succeed despite all the difficulties that may interfere: rules not disclosed because of lack of trust; credentials not found because their repository is unknown. What kind of properties of the policy protection policy and of the *hints* (see Section 4.4) guarantee a successful termination when the policy "theoretically" permits access to a resource?

- Optimal negotiations: which strategies optimize information disclosure during negotiation? Can reasonable preconditions prevent unnecessary information disclosure?
- In the presence of multiple ways of fulfilling a request, how should the client choose a response? We need both a language for expressing preferences, and efficient algorithms for solving the corresponding optimization problem. While this negotiation step is more or less explicitly assumed by most approaches on trust negotiation, there is no concrete proposal so far.

Additionally, integration of abductive semantics and ECA semantics is an open issue, as we have pointed out in a previous section.

4.5 Cooperative policy enforcement

Cooperative enforcement involves both machine-to-machine and human-machine aspects. The former is handled by negotiation mechanisms: published policies, provisional actions, hints, and other metalevel information (see Section 4.4) can be interpreted by the client to identify what information is needed to access a resource, and how to obtain that information.

Let us discuss the human-machine interaction aspect in more detail: One of the most important causes of the enormous number of computer security violations on the Internet is the users' lack of technical expertise. Users are typically not aware of the security policies applied by their system, neither of course about how those policies can be changed and how they might be improved by tailoring them to specific needs. As a consequence, most users ignore their computer's vulnerabilities and the corresponding countermeasures, so the system's protection facilities cannot be effectively exploited.

It is well known that the default, generic policies that come with system installations – often biased toward functionality rather than protection – are significantly less secure than a policy specialized to a specific context, but very few users know how to tune or replace the default policy. Moreover, users frequently do not understand what the policy really checks, and hence are unaware of the risks involved in many common operations.

Similar problems affect privacy protection. In trust negotiation, credential release policies are meant to achieve a satisfactory tradeoff between privacy and functionality – many interesting services cannot be obtained without releasing some information about the user. However, we cannot expect such techniques to be effective unless users are able to understand and possibly personalize the privacy policy enforced by their system.

A better understanding of a web service's policy makes it also easier for a first-time user to interact with the service. If denied access results simply in a "*no*" answer, the user has no clue on how he or she can possibly acquire the permission to get the desired service (e.g., by completing a registration procedure, by supplying more credentials or by filling in some form). This is why we advocate *cooperative policy enforcement*, where negative responses are enriched with suggestions and other explanations whenever such information does not violate confidentiality (sometimes, part of the policy itself is sensitive).

For these reasons, *greater user awareness and control on policies* is one of our main objectives, making policies easier to understand and formulate to the common user in the following ways:

- Adopt a *rule-based policy specification language*, because these languages are flexible and at the same time structurally similar to the way in which policies are expressed by nontechnical users.
- Make the policy specification language more friendly by e.g. developing a *controlled natural language* front-end to translate natural language text into executable rules (see next section).
- Develop *advanced explanation mechanisms* [24, 54, 55] to help the user understand what policies prescribe and control.

Inference Web (IW) [54, 55] is a toolkit that aims at providing useful explanations for the behavior of (Semantic-) Web based systems. In particular, [54] propose support for knowledge provenance information using metadata (e.g., Dublin Core information) about the distributed information systems involved in a particular reasoning task. [54] also deals with the issue of representing heterogeneous reasoning approaches, domain description languages and proof representations; the latter issue is addressed by using PML, the OWL-based Proof Markup Language [56].

Specifically applied to policies, [24] contains a requirements analysis for explanations in the context of automated trust negotiation and defines explanation mechanisms for *why, why-not, how-to*, and *what-if* queries. Several novel aspects are described:

- Adoption of a *tabled explanation structure* as opposed to more traditional approaches based on single derivations or proof trees. The tabled approach makes it possible to describe infinite failures, which is essential for *why not* queries.
- Explanations show simultaneously different possible proof attempts and allow users to see both local and global proof details at the same time. This combination of local and global (intra-proof and inter-proof) information facilitates navigation across the explanation structures.
- Introduction of suitable heuristics for focussing explanations by removing irrelevant parts of the proof attempts. A second level of explanations can recover missing details, if desired.
- Heuristics are *generic*, i.e. domain independent, they require no manual configuration.
- The combination of tabling techniques and heuristics yields a novel method for explaining failure.

Explanation mechanisms should be *lightweight* and *scalable* in the sense that (i) they do not require any major effort when the general framework is instantiated in a specific application domain, and (ii) most of the computational effort can be delegated to the clients. Queries are answered using the same policy specifications used for negotiation. Query answering is conceived for the following categories of users:

- Users who try to understand how to obtain access permissions;
- Users who monitor and verify their own privacy policy;
- Policy managers who verify and monitor their policies.

Currently, advanced queries comprise *why/why not, how-to*, and *what-if* queries. Why/why not queries can be used by security managers to understand why some specific request has been accepted or rejected, which may be useful for debugging purposes. Why-not queries may help a user to understand what needs to be done

in order to obtain the required permissions, a process that in general may include a combination of automated and manual actions. Such features are absolutely essential to enforce security requirements without discouraging users that try to connect to a web service for the first time. How-to queries have a similar role, and differ from why-not queries mainly because the former do not assume a previous query as a context, while the latter do.

What-if queries are hypothetical queries that allow to predict the behavior of a policy before credentials are actually searched for and before a request is actually submitted. What-if queries are good both for validation purposes and for helping users in obtaining permissions.

Among the technical challenges related to explanations, we mention:

- Find the right tradeoff between explanation quality and the effort for instantiating the framework in new application domains. Second generation explanation systems [57, 58, 59] prescribe a sequence of expensive steps, including the creation of an independent domain knowledge base expressly for communicating with the user. This would be a serious obstacle to the applicability of the framework.

Natural language policies

Policies should be written by and understandable to users, to let them control behavior of their system. Otherwise the risk that users keep on adopting generic hence ineffective built-in policies, and remain unaware of which controls are actually made by the system is extremely high – and this significantly reduces the benefits of a flexible policy framework.

Most users have no specific training in programming nor in formal logics. Fortunately, they spontaneously tend to formulate policies as rules; still, logical languages may be intimidating. For this reason, the design of front ends based on graphical formalisms as well as *natural language interfaces* are crucial to the adoption of formal policy languages. We want policy rules to be formulated like: "*Academic users can download the files in folder historical_data whenever their creation date precedes 1942*".

Clearly, the inherent ambiguity of natural language is incompatible with the precision needed by security and privacy specifications. Solutions to that can be the adoption of a *controlled* fragment of English (e.g., the ATTEMPTO system[6]) where a few simple rules determine a unique meaning for each sentence. This approach can be complemented with a suitable interface that clarifies what the machine understands.

5 Conclusions

Policies are really knowledge bases: a single body of declarative rules used in many possible ways, for negotiations, query answering, and other forms of system behavior control. As far as trust negotiation is concerned, we further argue that transparent interoperation based on ontology sharing can become "everyday technology" in a

[6] `http://www.ifi.unizh.ch/attempto/`

short time, and trust negotiation especially will become a success story for semantic web ideas and techniques.

In addition to stateless negotiation [60], we need stateful negotiation as well [45]. Even the Web, which started as a stateless protocol, now implements a number of techniques to simulate stateful protocols and transactions, especially in applications for accessing data other than web pages.

Cooperative policy enforcement and trust management gives common users better understanding and control on the policies that govern their systems and the services they interact with. The closer we get to this objective, the higher the impact of our techniques and ideas will be.

Policies will have to handle decisions under a wide range of risk levels, performance requirements, and traffic patterns. It is good to know that the rule-based techniques that different research communities are currently converging to are powerful enough to effectively address such a wide spectrum of scenarios. This is the level of flexibility needed by the Semantic Web.

About This Manuscript

This manuscript provides an introduction to policy representation and reasoning for the Semantic Web. It describes the benefits of using policies and presents four of the most relevant policy languages. 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 not anymore 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 harmonisation algorithms).

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

References

1. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, may 2001.
2. Grigoris Antoniou, Matteo Baldoni, Piero A. Bonatti, Wolfgang Nejdl, and Daniel Olmedilla. Rule-based policy specification. In Ting Yu and Sushil Jajodia, editors, *Secure Data Management in Decentralized Systems*, volume 33 of *Advances in Information Security*. Springer, 2007.

3. Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In *Security Protocols, 6th International Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63, Cambridge, April, 1998. Springer.
4. Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance checking in the policymaker trust management system. In *Financial Cryptography, Second International Conference*, volume 1465 of *Lecture Notes in Computer Science*, pages 254–274, Anguilla, British West Indies, February 1998. Springer.
5. Andrzej Uszok, Jeffrey M. Bradshaw, Renia Jeffers, Niranjan Suri, Patrick J. Hayes, Maggie R. Breedy, Larry Bunch, Matt Johnson, Shriniwas Kulkarni, and James Lott. KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *POLICY*, page 93, 2003.
6. Lalana Kagal, Timothy W. Finin, and Anupam 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*, Lecture Notes in Computer Science, pages 402–418. Springer, 2003.
7. Rita Gavriloaie, Wolfgang Nejdl, Daniel Olmedilla, Kent E. Seamons, and Marianne 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 *Lecture Notes in Computer Science*, pages 342–356, Heraklion, Crete, Greece, May 2004. Springer.
8. Piero A. Bonatti and Daniel 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, June 2005. IEEE Computer Society.
9. Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej 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.
10. Lalana Kagal, Massimo Paolucci, Naveen Srinivasan, Grit Denker, Timothy W. Finin, and Katia P. Sycara. Authorization and privacy for semantic web services. *IEEE Intelligent Systems*, 19(4):50–56, 2004.
11. 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.
12. William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. DARPA Information Survivability Conference and Exposition, IEEE Press, Jan 2000.
13. Wolfgang Nejdl, Daniel Olmedilla, Marianne Winslett, and Charles C. Zhang. Ontology-based policy specification and management. In *2nd European Semantic Web Conference (ESWC)*, volume 3532 of *Lecture Notes in Computer Science*, pages 290–302, Heraklion, Crete, Greece, May 2005. Springer.
14. Piero A. Bonatti, Claudiu Duma, Norbert Fuchs, Wolfgang Nejdl, Daniel Olmedilla, Joachim Peer, and Nahid Shahmehri. Semantic web policies - a discussion of requirements and research issues. In *3rd European Semantic Web Conference (ESWC)*, volume 4011 of *Lecture Notes in Computer Science*, Budva, Montenegro, June 2006. Springer.

15. Daniel Olmedilla. Security and privacy on the semantic web. In Milan Petkovic and Willem Jonker, editors, *Security, Privacy and Trust in Modern Data Management*. Springer, 2007 (to appear).
16. Jeffrey M. Bradshaw, Andrzej Uszok, Renia Jeffers, Niranjan Suri, Patrick J. Hayes, Mark H. Burstein, Alessandro Acquisti, Brett Benyo, Maggie R. Breedy, Marco M. Carvalho, David J. Diller, Matt Johnson, Shriniwas Kulkarni, James Lott, Maarten Sierhuis, and Ron 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)*, pages 835–842, Melbourne, Victoria, Australia, jul 2003. ACM.
17. Mike Dean and Guus Schreiber. OWL web ontology language reference, 2004.
18. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
19. Lalana Kagal. *A Policy-Based Approach to Governing Autonomous Behaviour in Distributed Environments*. PhD thesis, University of Maryland Baltimore County, 2004.
20. 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.
21. 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.
22. Jim Trevor and Dan 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.
23. Miguel Alves, Carlos Viegas Damásio, Wolfgang Nejdl, and Daniel Olmedilla. A distributed tabling algorithm for rule based policy systems. In *7th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2006)*, pages 123–132, London, Ontario, Canada, June 2006. IEEE Computer Society.
24. Piero A. Bonatti, Daniel Olmedilla, and Joachim Peer. Advanced policy explanations on the web. In *17th European Conference on Artificial Intelligence (ECAI 2006)*, pages 200–204, Riva del Garda, Italy, Aug-Sep 2006. IOS Press.
25. Pranam Kolari, Li Ding, Shashidhara Ganjugunte, Anupam Joshi, Timothy W. Finin, and Lalana 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, jun 2005. IEEE Computer Society.
26. Steffen Staab, Bharat K. Bhargava, Leszek Lilien, Arnon Rosenthal, Marianne Winslett, Morris Sloman, Tharam S. Dillon, Elizabeth Chang, Farookh Khadeer Hussain, Wolfgang Nejdl, Daniel Olmedilla, and Vipul Kashyap. The pudding of trust. *IEEE Intelligent Systems*, 19(5):74–88, 2004.
27. Grit Denker, Lalana Kagal, Timothy W. Finin, Massimo Paolucci, and Katia 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*, Lecture Notes in Computer Science, pages 335–350. Springer, 2003.
28. Daniel Olmedilla, Rubén Lara, Axel Polleres, and Holger Lausen. Trust negotiation for semantic web services. In *1st International Workshop on Semantic*

Web Services and Web Process Composition (SWSWPC), volume 3387 of *Lecture Notes in Computer Science*, pages 81–95, San Diego, CA, USA, July 2004. Springer.
29. Grid Security Infrastructure. http://www.globus.org/security/overview.html.
30. Andrzej Uszok, Jeffrey M. Bradshaw, and Renia 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*, Lecture Notes in Computer Science, pages 16–26. Springer, 2004.
31. Ionut Constandache, Daniel Olmedilla, and Wolfgang 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.
32. Li Gong. *Inside Java 2 Platform Security: Architecture, API Design, and Implementation*. Addison-Wesley, 1999.
33. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the 5th ICLP*, pages 1070–1080. MIT Press, 1988.
34. Elisa Bertino, Elena Ferrari, Francesco Buccafurri, and Pasquale Rullo. A logical framework for reasoning on data access control policies. In *Proc. of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, pages 175–189. IEEE Computer Society, 1999.
35. D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206–214, 1989.
36. Luigi Palopoli and Carlo Zaniolo. Polynomial-time computable stable models. *Ann. Math. Artif. Intell.*, 17(3-4):261–290, 1996.
37. Domenico Saccà and Carlo Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proc. of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'90)*, pages 205–217, 1990.
38. Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming.*, pages 89–148. Morgan Kaufmann, 1988.
39. P. A. Bonatti, N. Shahmehri, C. Duma, D. Olmedilla, W. Nejdl, M. Baldoni, C. Baroglio, A. Martelli, V. Patti, P. Coraggio, G. Antoniou, J. Peer, and N. E. Fuchs. Rule-based policy specification: State of the art and future work. Technical report, Working Group I2, EU NoE REWERSE, August 2004. `http://rewerse.net/deliverables/i2-d1.pdf`.
40. V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, J.J. Lu, A. Rajput, T.J. Rogers, R. Ross, and C. Ward. Hermes: Heterogeneous reasoning and mediator system. http://www.cs.umd.edu/projects/publications/ abstracts/hermes. html.
41. V. S. Subrahmanian, Piero A. Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert Ross. *Heterogenous Active Agents*. MIT Press, 2000.
42. Arnon Rosenthal and Marianne Winslett. Security of shared data in large systems: State of the art and research directions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 962–964. ACM, 2004.
43. Piero A. Bonatti, Claudiu Duma, Daniel Olmedilla, and Nahid Shahmehri. An integration of reputation-based and policy-based trust management. In *Seman-*

tic Web Policy Workshop in conjunction with 4th International Semantic Web Conference, Galway, Ireland, November 2005.
44. M. Blaze, J. Feigenbaum, and M. Strauss. Compliance Checking in the PolicyMaker Trust Management System. In *Financial Cryptography*, British West Indies, February 1998.
45. P.A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security*, 10(3):241–272, 2002. Short version in the Proc. of the Conference on Computer and Communications Security (CCS'00), Athens, 2000.
46. W. Winsborough, K. Seamons, and V. Jones. Negotiating Disclosure of Sensitive Credentials. In *Second Conference on Security in Communication Networks*, Amalfi, Italy, September 1999.
47. W. Winsborough, K. Seamons, and V. Jones. Automated Trust Negotiation. In *DARPA Information Survivability Conference and Exposition*, Hilton Head Island, SC, January 2000.
48. Marianne Winslett, Ting Yu, Kent E. Seamons, Adam Hess, Jared Jacobson, Ryan Jarvis, Bryan Smith, and Lina Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, 2002.
49. Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 6(1):1–42, 2003.
50. 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*, Yorktown Heights, June 2004.
51. K. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobsen, H. Mills, and L. Yu. Requirements for Policy Languages for Trust Negotiation. In *3rd International Workshop on Policies for Distributed Systems and Networks*, Monterey, CA, June 2002.
52. N. Li, W. Winsborough, and J.C. Mitchell. Distributed Credential Chain Discovery in Trust Management (Extended Abstract). In *ACM Conference on Computer and Communications Security*, Philadelphia, Pennsylvania, November 2001.
53. C. Zhang, P.A. Bonatti, and M. Winslett. Peeraccess: A logic for distributed authorization. In *12th ACM Conference on Computer and Communication Security (CCS 2005)*, Alexandria, VA, USA, 2005. ACM.
54. Deborah L. McGuinness and Paulo Pinheiro da Silva. Explaining answers from the semantic web: The inference web approach. *Journal of Web Semantics*, 1(4):397–413, 2004.
55. Deborah L. McGuinness and Paulo Pinheiro da Silva. Trusting answers from web applications. In *New Directions in Question Answering*, pages 275–286, 2004.
56. Paulo P. da Silva, Deborah L. McGuinness, and Richard Fikes. A proof markup language for semantic web services. Technical Report KSL Tech Report KSL-04-01, January, 2004.
57. William Swartout, Cecile Paris, and Johanna Moore. Explanations in knowledge systems: Design for explainable expert systems. *IEEE Expert: Intelligent Systems and Their Applications*, 6(3):58–64, 1991.
58. Michael C. Tanner and Anne M. Keuneke. Explanations in knowledge systems: The roles of the task structure and domain functional models. *IEEE Expert: Intelligent Systems and Their Applications*, 6(3):50–57, 1991.

59. M. R. Wick. Second generation expert system explanation. In J.-M. David, J.-P. Krivine, and R. Simmons, editors, *Second Generation Expert Systems*, pages 614–640. Springer Verlag, 1993.
60. Vladimir Kolovski, Yarden Katz, James Hendler, Daniel Weitzner, and Tim Berners-Lee. Towards a policy-aware web. In *Semantic Web Policy Workshop in conjunction with 4th International Semantic Web Conference*, Galway, Ireland, nov 2005.