PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web

Wolfgang Nejdl
L3S and
University of Hannover
Germany

nejdl@learninglab.de

Daniel Olmedilla
L3S and
University of Hannover
Germany

olmedilla@learninglab.de

Marianne Winslett
University of Illinois at
Urbana-Champaign
USA

winslett@cs.uiuc.edu

ABSTRACT

Researchers have recently begun to develop and investigate policy languages to describe trust and security requirements on the Semantic Web [14, 24]. Such policies will be one component of a run-time system that can negotiate to establish trust on the Semantic Web. In this paper, we show how to express different kinds of access control policies and control their use at run time using PeerTrust, a new approach to trust establishment. We show how to use guarded distributed logic programs as the basis for PeerTrust's simple yet expressive policy and trust negotiation language, built upon the rule layer of the Semantic Web layer cake. We describe the syntax and semantics of GDLPs, and compare PeerTrust's language to other approaches to implementing policies and trust negotiation. Through examples, we show how PeerTrust can be used to support delegation, policy protection and negotiation strategies. Finally, we discuss the PeerTrust automated trust negotiation engine prototype implemented in Prolog, and identify areas for further research.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: Miscellaneous; I.2.4 [**Knowledge Representation Formalisms and Methods**]: Representation languages, Representations (procedural and rule-based); H.2 [**Database Management**]: General—*Security, Integrity, and Protection*

General Terms

Design, Languages, Management, Security, Algorithms

Keywords

Automated Trust Negotiation, Peer-to-Peer, Semantic Web, Security, Policy Languages

1. INTRODUCTION

As peer-to-peer architectures start to move into use for applications based on the Semantic Web, they must address the issue of access control for sensitive resources provided by peers in the network [14, 24], such as services, documents, roles, and capabilities. For example, in the Edutella infrastructure [19, 18, 20], each peer manages distributed resources described by RDF metadata, and interfaces to the Edutella network using a Datalog-based query language. The early Edutella testbeds focussed on providing distributed learning repositories in an environment where all resources are freely available; the main research focus was efficient searching for course-related information using appropriate queries over the metadata available for that information. More recently, however, the Edutella infrastructure has been deployed in the context of the EU/IST ELENA project [23], whose participants include e-learning and e-training companies, learning technology providers, and several universities and research institutes (see also http://www.elena-project.org/). To meet the needs for access control in this peer-to-peer network that connects commercial e-learning providers and learning management systems, Edutella must also support access control policies that describe who is allowed to access each document and service.

For example, suppose that E-Learn Associates manages a Spanish course in the peer-to-peer network, and Alice wishes to access the course. If the course is accessible free of charge to all police officers who live in and work for the state of California, Alice can show E-Learn her digital police badge to prove that she is a state police officer, as well as her California driver's license, and subsequently can gain access to the course at no charge.

However, Alice may not feel comfortable showing her police badge to just anyone; she knows that there are Web sites on the west coast that publish the names, home addresses, and home phone numbers of police officers. We can view her police badge as an item on the Semantic Web, protected by its own access control policy. For example, Alice may only be willing to show her badge to companies that belong to the Better Business Bureau of the Internet. But with the introduction of this additional policy, access control is no longer the one-shot, unilateral affair that one finds in traditional distributed systems or in recent proposals for access control on the Semantic Web [14, 24]: in order to see an appropriate subset of Alice's digital credentials, E-Learn will have to show that it satisfies the access control policies for each of them; and in the process of demonstrating that it satisfies those policies, it may have to disclose additional credentials of its own, but only after Alice demonstrates that she satisfies the access control policies for each of *them*; and so on. Thus the use of policies and digital credentials as a basis for access control on the semantic web raises a number of challenging run-time issues:

- How can Alice and E-Learn find out about each other's relevant access control policies, so that they can prove that they satisfy them?

- Given that there may be many ways that Alice can prove that she satisfies a particular policy of E-Learn's (by disclosing different subsets of her credentials), how can she decide

which subset to disclose?

- Often Alice may not have in her possession all the credentials she needs to satisfy one of E-Learn's policies. For example, E-Learn may offer a discounted price for its French course if Alice can demonstrate that she is a student at an accredited university. Alice probably has her student ID in hand, but how can she automatically collect the necessary credentials to show that her university is accredited?

- Traditional distributed systems security solutions (e.g., Kerberos) are centralized, which runs counter to the autonomous, peer-to-peer nature of the Semantic Web. How can we meet all the above goals without resorting to a centralized approach, while still guaranteeing individual autonomy to the extent possible and simultaneously guaranteeing that Alice and E-Learn will be able to establish trust—i.e., that Alice will be able to access E-Learn's courses—if at all possible?

In this paper, we build upon the previous work on policy-based access control for the Semantic Web by showing how to use *automated trust negotiation* to answer these questions, as embodied in the PeerTrust approach to access control. We start by introducing the concepts behind trust negotiation in section 2. We then introduce guarded distributed logic programs to express and implement trust negotiation in a distributed environment, in section 3. We discuss PeerTrust's trust negotiation using guarded distributed logic programs in detail in section 4, and describe an evaluation algorithm as well as as a prototype PeerTrust implementation in section 5. We discuss related work in section 6 and conclude with a brief look at further research issues.

2. TRUST NEGOTIATION

In traditional distributed environments, service providers and requesters are usually known to each other. Often shared information in the environment tells which parties can provide what kind of services and which parties are entitled to make use of those services. Thus, trust between parties is a straightforward matter. Even if on some occasions there is a trust issue, as in traditional client-server systems, the question is whether the server should trust the client, and not vice versa. In this case, trust establishment is often handled by uni-directional access control methods, such as having the client log in as a pre-registered user.

One of the significant differences between the Semantic Web and traditional distributed systems is that the Semantic Web provides an environment where parties may make connections and interact without being previously known to each other. In many cases, before any meaningful interaction starts, a certain level of trust must be established from scratch. Generally, trust is established through exchange of information between the two parties. Since neither party is known to the other, this trust establishment process should be bi-directional: both parties may have sensitive information that they are reluctant to disclose until the other party has proved to be trustworthy at a certain level. As there are more service providers emerging on the Web every day, and people are performing more sensitive transactions (for example, financial and health services) via the Internet, this need for building mutual trust will become more and more common.

Currently on the Internet, the prevailing authorization approach is still the traditional identity-based access control method, where clients are required to pre-register with a server, in order to obtain a local login, capability, or identity certificate before requesting that service. There are several disadvantages of this approach.

1. Such practices are usually offered in a "take-it-or-leave-it" fashion, i.e., the clients either *unconditionally* disclose their information to the server in the pre-registration phase, or do not get the service at all. There is no chance for the clients to apply their own access control policies for their information, and decide accordingly whether the server is trustworthy enough so that sensitive information can be disclosed.

2. When the clients fill out the online registration form, it is very hard for the server to verify whether the provided information is valid.

3. Often, much of the information required in online registration forms does not seem directly related to the service that clients want to access. However, since the clients have to unconditionally provide the information, there is no way that they can discuss their misgivings with the server.

A particularly important example of the latter shortcoming is that the traditional access control methods usually require the customer to reveal her exact identity, which, in many cases, is irrelevant to the service the customer requests. For example, when Alice enrolls in the free Spanish course, it suffices for her to prove that she is a police officer. Other personal information, such as her name, her home phone number, and her address, does not need to be revealed.

Digital credentials [1] (or simply *credentials*) make it feasible to manage trust establishment efficiently and bi-directionally on the Internet. Digital credentials are the on-line counterparts of paper credentials that people use in their daily life, such as a driver's license. By showing appropriate credentials to each other, a service requester and provider can both prove their qualifications. In detail, a credential is a digitally signed assertion by the *credential issuer* about the properties of one or more entities. The credential issuer uses its private key to sign the credential, which describes one or more attributes and/or relationships of the entities. The public key of the issuer can be used to verify that the credential was actually issued by the issuer, making the credential verifiable and unforgeable. The signed credential can also include the public keys of the entities referred to in the credential. This allows an entity to use her private key to prove that she is one of the entities referred to in the credential [21]. In practice, digital credentials can be implemented in many ways, including X.509 [12] certificates, anonymous credentials [3, 5, 4], and signed XML statements [8].

When credentials do not contain sensitive information, the trust establishment procedure is very simple. For example, if Alice will show her police badge and driver's license to anyone, then she (more precisely, a software agent acting on her behalf) first checks E-Learn's policy for its Spanish course, which is always available. Then she presents her police badge and license along with the retrieval request. However, in many situations, especially in the context of e-business, credentials themselves contain sensitive information. For example, as discussed earlier, Alice may only be willing to show her police badge to members of the Better Business Bureau. To deal with such scenarios, a more complex procedure needs to be adopted to establish trust through negotiation.

In the PeerTrust approach to automated trust establishment, trust is established gradually by disclosing credentials and requests for credentials, an iterative process known as *trust negotiation*. This differs from traditional identity-based access control systems mainly in the following aspects:

1. Trust between two strangers is established based on parties' properties, which are proven through disclosure of digital credentials.

2. Every party can define access control policies to control outsiders' access to their sensitive resources. These resources can include services accessible over the Internet, documents and other data, roles in role-based access control systems, credentials, policies, and capabilities in capability-based systems.

3. In the approaches to trust negotiation developed so far, two parties establish trust directly without involving trusted third parties, other than credential issuers. Since both parties have access control policies, trust negotiation is appropriate for deployment in a peer-to-peer architecture, where a client and server are treated equally. Instead of a one-shot authorization and authentication, trust is established incrementally through a sequence of bilateral credential disclosures.

A trust negotiation is triggered when one party requests to access a resource owned by another party. The goal of a trust negotiation is to find a sequence of credentials $(C_1, \ldots, C_k, R)$, where R is the resource to which access was originally requested, such that when credential C_i is disclosed, its access control policy has been satisfied by credentials disclosed earlier in the sequence—or to determine that no such credential disclosure sequence exists. (For uniformity of terminology, we will say that R is *disclosed* when E-Learn grants Alice access to R.)

The following online transaction example shows what the trust negotiation process may look like when Alice registers for the free Spanish course.

Step 1 Alice requests to access E-Learn's free Spanish course.

Step 2 E-Learn replies by requesting Alice to show a police badge issued by the California State Police to prove that she is a police officer, and her driver's license to prove that she is living in the state of California.

Step 3 Alice is willing to disclose her driver's license to anyone, so she sends it to E-Learn. However, she considers her police badge to contain sensitive information. She tells E-Learn that in order to see her police badge, E-Learn must prove that it belongs to the Better Business Bureau.

Step 4 Fortunately, E-Learn does have a Better Business Bureau membership card. The card contains no sensitive information, so E-Learn discloses it to Alice.

Step 5 Alice now believes that she can trust E-Learn and discloses her police badge to E-Learn.

Step 6 After verifying that the badge is valid and that Alice owns it and the driver's license, E-Learn gives Alice the special discount for this transaction.

In practice, the above process is conducted by security agents who interact with each other on behalf of users. A user only needs to specify access control policies for credentials and other resources. The actual trust negotiation process is fully automated and transparent to users. Further, the above example used objective criteria for determining whether to allow the requested access. More subjective criteria, such as ratings from a local or remote reputation monitoring service, can also be included in a policy.

In the remainder of this paper we will show how to specify and apply access control policies and trust negotiation using guarded distributed logic programs, building on the rule layer of the Semantic Web. Before we delve into details, though, let us highlight two general criteria for trust negotiation languages as well as two important features already mentioned briefly above. A more detailed discussion can be found in [22].

Well-defined semantics. Two parties must be able to agree on whether a particular set of credentials in a particular environment satisfies a policy. To enable this agreement, a policy language needs a clear, well-understood semantics.

Expression of complex conditions. A policy language for use in trust negotiation needs the expressive power of a simple query language, such as relational algebra plus transitive closure. Such a language allows one to restrict attribute values (e.g., age must be over 21) and relate values occurring in different credentials (e.g., the issuer of the student ID must be a university that ABET has listed as accredited).

Sensitive policies. The information in an access control policy can reveal a lot about the resource that it protects. For example, who is allowed to see Alice's medical record? Her parole officer? Her psychiatrist or social worker? Similarly, a policy for access to a web page giving information about a secret joint venture might immediately reveal which companies were participating in the venture. Because policies can contain sensitive information, and because they may be shown to outsiders, they need to be protected like any other shared resource. Previous work on trust negotiation has looked at a variety of ways of protecting the information in policies. In this paper, we will use the protection scheme introduced in [28], which gives (opaque) names to policies and allows any named policy $P1$ to have its own access control policy $P2$, meaning that the contents of $P1$ can only be disclosed to parties who have shown that they satisfy $P2$. To give flexibility in assigning different levels of protection to different aspects of a policy, UniPro also allows the definition of a policy P to refer to other policy definitions by name.

Delegation. Trust negotiation research has also addressed the issue of delegation of authority. For example, rather than issuing student IDs directly, a university may delegate that authority to its registrar. Then student IDs from that university will not bear the digital signature of the university itself, but rather the signature of the registrar. To prove that Bob is a student at UIUC, then, he will have to present both his student ID and the (signed) policy from UIUC that delegates authority to the registrar to issue IDs. This level of detail will not be present in E-Learn's policy for giving student discounts, which will simply say that Bob has to be a student at UIUC. If E-Learn's policy says that Bob must be a student at an institution accredited by ABET, Bob faces additional challenges during negotiation: how can he find the credentials that show that his university is accredited, or conclude that no such credentials exist? Previous work on trust negotiation has addressed the questions of how to specify and reason about delegations of authority [15] and how to find credentials [16].

3. GUARDED DISTRIBUTED LOGIC PROGRAMS

3.1 Syntax

3.1.1 Definite Horn Clauses

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 are the basis for logic programs [17], which have been used as the basis for the rule layer of the Semantic Web and specified in the RuleML effort ([6, 7]) as well as in the recent OWL Rules Draft [11]. 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, we will not dwell on them in this paper, but rather focus on other more unusual required language extensions.

Definite Horn clauses are used in the Edutella infrastructure to represent each peer's knowledge about its local resources, including services, data, credentials, and the access control policies for its resources. Edutella also uses a restricted form of definite Horn clauses as the language peers use to query one another, as well as the language used to represent query answers. This language is a strict superset of relational algebra.

On top of this definite Horn clause language, we need to add some additional features, discussed in the next sections.

3.1.2 Access to Security-Related Libraries

Trust negotiation relies on the use of Public Key Infrastructure (PKI). This term is often used to specifically refer to X.509 [12], but we rely on PKI only in a broader sense: a public key infrastructure is a framework specifying how digital credentials are created, distributed, stored and revoked. The best-known approaches to PKI are X.509, the web-of-trust approach of PGP [29], and anonymous credential systems (also called pseudonym systems) [3, 4, 5]. Trust negotiation can use credentials from any or all of these approaches.

With a PGP- or X.509-based approach to trust negotiation, an external function call must be made at run time to verify the contents of each received credential by determining whether the credential really was signed by its supposed issuer. This may involve looking up the public key of the issuer at a trusted third party. Further, when a policy specifies that the requester must be a specific person mentioned in a credential (for example, that Alice must be the police officer whose name appears on the police badge that she has disclosed), then an external function call to an authentication library is needed so that Alice can prove that she is that person, generally by interactively proving that she possesses the private key associated with the public key that in turn is associated with that particular field of the credential. Because Alice will simultaneously have many identities (e.g., employee number 983745234 on her police badge, state ID number C234284234WG on her driver's license, and session ID number 3312 in her conversation with eOrg), Alice must provide a separate proof of possession for each identity she uses in the credentials she submits. In an anonymous credential system, external function calls can be used, for example, to invoke a routine that interactively creates a zero-knowledge proof that a particular policy is satisfied.

The literals in definite Horn clauses can represent external procedure calls (evaluating either to true or false). We can use these to call authentication libraries and check environmental conditions that are mentioned in a policy, such as the time and date. For conciseness, the policies that we present in this paper do not include these calls, and are written as though Alice (and every other party) has a single global identity, "Alice", and she has already convinced eOrg that she is in fact "Alice".

3.1.3 References to Other Peers

The ability to reason about statements made by other peers is central to trust negotiation. For example, in section 2, E-Learn wants to see a statement from Alice's employer that says that she is a police officer. One can think of this as a case of E-Learn *delegating evaluation* of the query "Is Alice a police officer?" to the California State Police (CSP). Once CSP receives the query, the manner in which CSP handles it may depend on who asked the query. Thus CSP needs a way to specify which peer made each request that it receives.

To express delegation of evaluation to another peer, we extend each literal lit_i with an additional *Issuer* argument,

$$lit_i \text{ @ Issuer}$$

where *Issuer* specifies the peer who is responsible for evaluating lit_i or has the authority to evaluate lit_i. For example, E-Learn's discount policy might mention policeOfficer(alice) @ csp. If that literal evaluates to true, then CSP says that Alice is a California police officer.

As another example, a company eOrg may have a policy that students at UIUC are preferred customers.

eOrg:
preferred(X) ← student(X) @ uiuc.

This policy says that the UIUC peer is responsible for certifying the student status of a given person[1]. (For clarity, we prefix each rule by the peer in whose knowledge base it is included.)

The *Issuer* argument can be a nested term containing a sequence of issuers, which are then evaluated starting at the outermost layer. For example, UIUC is unlikely to be willing to answer eOrg's query about whether Alice is enrolled at UIUC. A more practical approach is for eOrg to ask Alice to evaluate the query herself, i.e., to send eOrg her student ID:

eOrg:
student(X) @ uiuc ←
  student(X) @ uiuc @ X.

As mentioned earlier, CSP and UIUC may need a way of referring to the peer who asked a particular query. This can be accomplished with a second *Requester* argument for literals, so that we now have literals of the form

$$lit_i \text{ @ Issuer \$ Requester}$$

For example, Alice may find itself evaluating the literal student(alice) @ uiuc \$ eOrg. The *Requester* argument can be nested, too, in which case it expresses a chain of requesters, with the most recent requester included in the outermost layer of this nested term.

Using the *Issuer* and *Requester* arguments, we can delegate evaluation of literals to other peers and also express interactions and the corresponding negotiation process between different peers. For example, consider E-Learn Associates' policy for free Spanish courses for California police officers:

eLearn:
freeEnroll(Course, Requester) \$ Requester ←
  policeOfficer(Requester) @ csp @ Requester,
  spanishCourse(Course).

---

[1] As a final reminder that authentication is a vital part of PeerTrust, we point out that in practice this policy must be written as preferred(X) ← student(Y) @ uiuc, authenticatesTo(X,Y), where authenticatesTo is an external predicate that allows Alice to prove at run time that she possesses the identity (i.e., the student ID number) under which she is known at UIUC.

If the user provides appropriate identification, then the policy for the free enrollment service is satisfied, and E-Learn will allow the user to access the service through a mechanism not shown here. In this example, the mechanism can transfer control directly to the enrollment service. For some services, the mechanism may instead give Alice a nontransferable token that she can use to access the service repeatedly without having to negotiate trust again until the token expires. The mechanism can also implement other security-related measures, such as creating an audit trail for the enrollment. When the policy for a negotiation-related resource such as a credential becomes satisfied, the run-time system may choose to include it directly in a message sent during the negotiation, as discussed later.

3.1.4   Local Rules and Signed Rules

Each peer defines an access control 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 in its proofs in certain situations.

For example, Alice can use a rule (with an empty body in this case) that was defined by UIUC to prove that she is really a UIUC student:

alice:
student(alice) @ uiuc
  signedBy [uiuc].

In this example, the "signedBy" term indicates that the rule has UIUC's digital signature on it. This is very important, as E-Learn is not going to take Alice's word that she is a student; she must present a statement signed by the university to convince E-Learn. A signed rule has an additional argument that says who issued the rule. The cryptographic signature itself is not included in the logic program, 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. We assume that when a peer receives a signed rule from another peer, the signature is verified before the rule is passed to the GDLP 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. For example, the UIUC registrar can use a signed rule to prove that it is entitled to determine who is a student at UIUC:

uiucRegistrar:
student(X) @ uiuc ←
  signedBy [uiuc]
 student(X) @ uiucRegistrar.

If Alice's student ID is signed by the registrar, then she should cache a copy of the rule given above and submit both the rule and the student ID when E-Learn asks her to prove that she is a UIUC student.

3.1.5   Guards

To guarantee that all relevant policies are satisfied before access is given to a resource, we must sometimes specify a partial evaluation order for the literals in the body of a rule. Similar to approaches to parallel logic programming such as Guarded Horn Logic [26], we split the body's literals into a sequence of sets, divided by the symbol "|". All but the last set are *guards*, and all the literals in one set must evaluate to true before any literals in the next set are evaluated. We will see examples of the need for guards later on.

Our guards express the evaluation precedence of literals within a rule. In contrast to most parallel logic programming systems employing guards, our guards do not introduce a deterministic choice of one single rule to evaluate next. Any rule whose guard literals evaluate to true can be used for continued evaluation. Both local and signed rules can include guards.

3.1.6   Public and Private Predicates

In trust negotiation, we must be able to distinguish between predicates that can be queried by external peers, and ones that cannot—analogous to the public and private procedures in object-oriented languages. For example, authentication and time-of-day predicates are private. Peers may also have private rules, which are neither shown to nor can directly be called by other peers. Public and private rules and predicates are straightforward to design and implement in definite Horn clauses. In the examples in this paper, we will specifically mark private rules as being private; all unmarked rules are public.

3.2   Semantics

The semantics of the PeerTrust language is an extension of that of SD3 [25]. We add additional arguments in each literal for the *Issuer* and *Requester* arguments, and add the notion of distributed *Peers* with their respective knowledge bases. In this global semantics, instead of having theories with the literals

$$P_j(t_1, \ldots, t_n) \text{ @ Issuer \$ Requester}$$

in each peer, we have

$$P_j^+(t_1, \ldots, t_n, \text{Issuer, Requester, Peer})$$

The result is a global logic program that views the set of all peer knowledge bases as one global knowledge base. The *Issuer* and *Requester* arguments can be nested and therefore include more than one value. Due to space constraints, we omit the details of how we handle the Issuer, Requester, and Peer arguments.

4.   AUTOMATED TRUST NEGOTIATION IN DETAIL: EXAMPLES AND GDLPS

We will now extend the PeerTrust examples presented informally in the preceding sections and show how to represent the appropriate policies and negotiation rules for automated trust negotiation using guarded distributed logic programs.

4.1   Scenario 1: Alice & E-Learn

E-Learn Associates sells learning resources and gives special offers to some users. For example, E-Learn clients can get a discount if they are preferred customers at the ELENA consortium. This is represented by the following two rules that govern access to the "discountEnroll" service:

eLearn:
discountEnroll(Course, Requester) \$ Requester ←
 eligibleForDiscount(Requester, Course).
eligibleForDiscount(X, Course) ← preferred(X) @ elena.

At run time, E-Learn could ask ELENA whether each E-Learn client is a preferred ELENA customer, but that is not necessary because ELENA has given E-Learn a signed rule specifying how ELENA computes the "preferred" status of individuals.

preferred(X) @ elena ←
  signedBy [elena]
 student(X) @ uiuc.

E-Learn could also ask the university directly about the student status of its customers, but that would involve the UIUC peer in a lot of trust negotiations that do not interest it. So instead E-Learn will ask students themselves to provide full proof of their student status, as expressed by the following rule:

student(X) @ University ←
  student(X) @ University @ X.

Furthermore, E-Learn is a member of the Better Business Bureau, and can prove it through an appropriate signed rule:

member(eLearn) @ bbb
  signedBy [bbb].

UIUC employs a registrar to whom it delegates student status certification. This is expressed by an appropriate delegation rule from UIUC to the UIUC registrar. This is UIUC's private rule, however, and UIUC does not directly respond to any queries about student status.

UIUC:
student(X) ←
  student(X) @ uiucRegistrar.

Students get a credential from the UIUC registrar certifying their student status, and another credential certifying the delegation from UIUC to the UIUC registrar. Alice has both of these credentials. Her policy is to give out her credentials only to members of the Better Business Bureau, and she asks them to produce a proof of this membership themselves. Her policy appears as a new level of guards in the signed rules. The new guards are outside the scope of the original signature, which we represent by placing the signature after the new guards and before the first (empty) level of literals in the rules. This syntactic sugar helps us remember exactly what was signed, as new guards can be added only at the beginning of the body of the rule.

alice:
student(X) @ uiuc $ Requester
  ← member(Requester) @ bBB @ Requester |
   signedBy [uiuc]
    student(X) @ uiucRegistrar.
student(alice) @ uiucRegistrar $ Requester
  ← member(Requester) @ bBB @ Requester |
   signedBy [uiucRegistrar] .

With the PeerTalk run-time system and this set of policies, Alice will be able to access the discounted enrollment service at E-Learn.

4.2 Scenario 2: Signing Up for Learning Services

The following scenario uses policies to control access to ELENA Web services, including course enrollment and delivery. In this scenario, Bob works for the HR department of IBM, and is in charge of buying new e-learning courses. He has the authority to buy courses costing up to $2000. He is only willing to disclose this authorization to ELENA members.

bob:
email(bob, bob@ibm.com).
employee(bob) @ ibm $ Requester
  ← elenaMember(Requester)
   | signedBy [ibm].
authorized(bob, Price) @ ibm $ Requester

  ← elenaMember(Requester)
   | signedBy[ibm]
   Price < 2000.
elenaMember(Requester) ←
  member(Requester) @ elena @ Requester.

Bob usually pays with his company's credit card, but will not even discuss the existence of the card with non-members of ELENA. Further, he will only disclose the card to ELENA members who are authorized by VISA to accept VISA cards. The credit card is signed by VISA and includes many fields; for conciseness here we show only a name field, containing "ibm".

visaCard(ibm) $ Requester
  ← elenaMember(Requester) | policy29(Requester)
   signedBy [visa].
policy29(Requester) ←
  authorizedMerchant(Requester) @ visa @ Requester.

Bob also knows that IBM and E-Learn are members of the ELENA consortium.

member(ibm) @ elena
  signedBy [elena].
member(eLearn) @ elena
  signedBy [elena].

E-Learn Associates offers free courses and pay-per-use courses. Free courses are available to employees of ELENA network members. Pay-per-use courses require an authorization from the company as well as the company's VISA information for billing. When a course is made available, a notification is sent to the requester, and the requester is billed if appropriate. Notification and billing are handled by an external mechanism, as always, and the "extra" variables in some rule heads are needed by those external functions.

eLearn:
enroll(Course, Requester, Company, Email, 0) $ Requester ←
  freeCourse(Course),
  freebieEligible(Course, Requester, Company, EMail).
enroll(Course, Requester, Company, Email, Price) $ Requester ←
  policy49(Course, Requester, Company, Price) |

The following rules express the policies for free and pay-per-use courses:

freebieEligible(Course, Requester, Company, EMail) ←
  email(Requester, EMail) @ Requester,
  employee(Requester) @ Company @ Requester,
  member(Company) @ elena @ Requester.
policy49(Course, Requester, Company, Price) ←
  price(Course, Price),
  authorized(Requester, Price) @ Company @ Requester,
  visaCard(Company) @ visa @ Requester.

The use of policy names in the above example allows us to protect policies in the same way as other resources. For example, E-Learn's partner agreements and customer list are privileged business information. Without additional protection, anyone can learn that E-Learn's only partner agreement that involves free course registration is with ELENA. To avoid disclosing this sensitive information during negotiation, one approach is to decree that the exact criteria for free course registration should only be disclosed to E-Learn employees. We can accomplish this by making freebieEligible a private rule and removing the Requester arguments from

its body literals, and introducing a new public version, freebieEligible2, that contains a new guard that checks whether the requester is an E-Learn employee. When the guard in freebieEligible2 is not satisfied, we will never query the requester about the other literals in freebieEligible2. ELENA member companies can disseminate copies of freebieEligible to their employees, so the employees know to push the appropriate credentials to E-Learn to satisfy the private rule and gain access to free courses.

eLearn:
freebieEligible2(Course, Requester, Company, EMail) ←
  policy3(Requester) |
  email(Requester, EMail) @ Requester,
  employee(Requester) @ Company @ Requester,
  member(Company) @ elena @ Requester.
policy3(Requester) ←
  employee(Requester) @ eLearn.

Finally, E-Learn has a database of course information, and may have cached other signed rules and credentials from other peers (e.g., to speed up negotiation, or for use in the private rule for determining eligibility for free course enrollment). For example:

freeCourse(cs101).
freeCourse(cs102).
price(cs411, 1000).
member(ibm) @ elena.
  signedBy [elena]
authorizedMerchant(eLearn)
  signedBy[visa].

To check if a requester's VISA card has been revoked, E-Learn must make an external function call to a VISA card revocation authority. (This approach provides a run-time semantics for the revocation speech acts mentioned in [14].) E-Learn can implement this as an extension of policy49, where E-Learn checks for credit card revocation directly with VISA and ensures that the purchase price will not cause the account balance to exceed its credit limit.

policy49(Course, Requester, Company, Price) ←
  price(Course, Price),
  authorized(Requester, Price) @ Company @ Requester,
  visaCard(Company) @ visa @ Requester,
  purchaseApproved(Company, Price) @ visa.

To help E-Learn decide where to send a particular query, it can keep a database listing authoritative peers for various topics. At run time, unbound *Issuer* arguments can be instantiated from this database. In this case E-Learn might have a list of authorities it can ask about specific predicates:

policy49(Course, Requester, Company, Price) ←
  price(Course, Price),
  authorized(Requester, Price) @ Company @ Requester,
  visaCard(Company) @ visa @ Requester,
  authority(purchaseApproved, Authority),
  purchaseApproved(Company, Price) @ Authority.

These lists of authorities can also come from a broker, as shown in the following modification of the previous example:

policy49(Course, Requester, Company, Price) ←
  price(Course, Price),
  authorized(Requester, Price) @ Company @ Requester,
  visaCard(Company) @ visa @ Requester,
  authority(purchaseApproved, Authority) @ myBroker,
  purchaseApproved(Company, Price) @ Authority.

With the PeerTalk run-time system and these policies, IBM employees will be able to enroll in free courses at E-Learn. If IBM were not a member of ELENA, then IBM employees would not be eligible for free courses, but Bob would be able to purchase courses for them from E-Learn.

Bob might want to delegate authority to another peer to carry out a negotiation on his behalf. For example, handheld devices may not have enough power to carry out trust negotiation directly. In this case, Bob's device can forward any queries it receives to a another peer that Bob trusts, such as his home or office computer. This trusted peer has access to Bob's policies and credentials, performs the negotiation on his behalf, and returns the final results to the handheld device. If desired, this can be implemented in a manner that allows Bob's private keys to reside only on his handheld device, to reduce the amount of trust that Bob must place in the other peers.

5. ALGORITHMS AND IMPLEMENTATION

5.1 A Simple Evaluation Algorithm

Guarded distributed logic programs can be evaluated in many different ways. This flexibility is important, as different networks of peers, and even different peers within the same network, may prefer different approaches to evaluation. In this paper, we will present an extremely simple evaluation algorithm for PeerTrust that is based on the cooperative distributed computation of a proof tree, with all peers employing the same evaluation algorithm. The algorithm assumes that each peer uses a queue to keep track of all active proof trees and the expandable subqueries in each of these trees. The proof trees contain all information needed during negotiation, including used rules, credentials and variable instantiations. Peers communicate with one another by sending queries and query answers to each other.

The following sketch of the algorithm uses EITHER:/OR: to express a non-deterministic choice between several possible branches of the algorithm.

Let $TreeList$ denote the structure with all active proof trees
Set $TreeList := []$
Let $Tree$ denote the structure holding $Query\$Requester$ and $Proof$
    both of which may still contain uninstantiated variables
Loop
    EITHER:
        Receive $Tree$: a query to answer / a goal to prove
        Add_New_Tree($Tree$, $TreeList$)
    OR:
        Receive $Answer(Tree)$
        Add_Answers($Answer(Tree)$, $TreeList$)
    OR:
        Receive $Failure(Tree)$ from peer
        Send $Failure(Tree)$ to Requester
        Remove_Tree($Failure(Tree)$, $TreeList$)
    OR:
        Process_a_Tree($TreeList$)
end Loop

At each step a peer can receive a new query from another peer, receive answers or learn that there are no answers for a query it previously sent to a peer, or selects one of its active trees for processing. If this tree is already complete, the answers can be returned to the peers who requested this evaluation. If the tree contains subqueries which still have to be evaluated, the peer selects one of them and tries to evaluate it.

```
Process_a_Tree(TreeList)
Let NewTrees denote the new proof trees
Set NewTrees := []
Select_Tree(Tree, TreeList, RestOfTreeList)
IF all subqueries in Tree are already evaluated
THEN
    Send (Answer(Tree)) to Requester
    TreeList : −RestTreeList
ELSE
    Select_Subquery (SubQuery,Tree)
    IF SubQuery can be evaluated locally
    THEN
        Loop while new local rules are found
            Expand SubQuery into its subgoals
            Update_Tree(Tree,NewSubgoals)
            Add_Tree(Tree,NewTrees)
        End loop
    ELSE //if it is a goal with an "@ Issuer" suffix,
        // indicating remote evaluation
        IF peer has Signed_Rule(SubQuery)
            Loop while new signed rules are found
                Expand SubQuery into its subgoals
                Update_Tree(Tree,NewSubgoals)
                Add_Tree(Tree,NewTrees)
            End loop
        ELSE
            Send Request(SubQuery) to Issuer)
            Update_Status(Tree, waiting)
        END IF
    END IF
    IF no local or remote expansion for SubQuery was possible
        Send (Failure(Tree)) to Requester
    ELSE
        Add_New_Trees
            (NewTrees,RestTreeList,NewTreeList)
    END IF
    TreeList : −NewTreeList
END IF
```


Figure 1: Meta-interpreter example

Expansion of subqueries is done either locally (using the peer's rules and signed rules) or by sending the subquery to a remote peer. Many queries per proof can be active (i.e., awaiting answers and being processed) at any time. Each new query from a remote peer starts a new proof tree while answers from remote peers are "plugged into" existing proof trees. An example of a query expansion in a proof tree is depicted in figure 1, where a tree is expanded into two and then three trees. Each tree structure contains at least its root and leaves, plus any additional information from the proof, including credentials, that we want to keep and/or return to the requester. If one proof tree for the original query is completed, then the negotiation is over and the requester obtains access to the desired resource.

This algorithm can be extended and improved in many different ways. For example, it can be made more efficient and effective at run time by generalizing the definition of a query, allowing iteration through a set of query answers, allowing intensional query answers, support for caching of query answers, and prioritization of rules a la [6]. Alternatively, the algorithm can be revamped in ways that will allow different peers to choose different evaluation algorithms, along the lines of [28], or to provide provable guarantees of completeness and termination, as offered by the algorithms of [28]. No matter what revisions are made, however, at its heart any evaluation algorithm will be working to construct a certified proof tree.
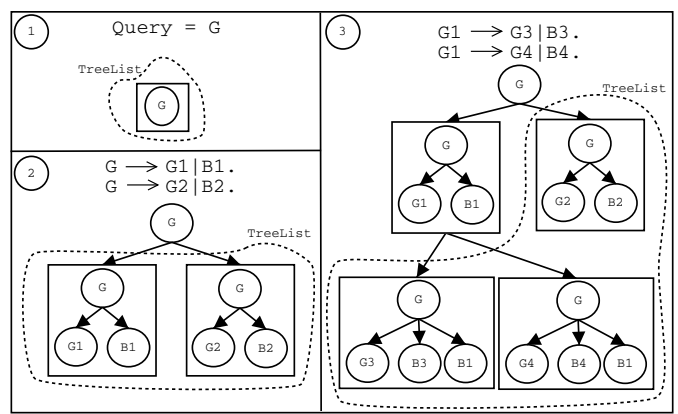
5.2 Prototype Implementation

As of mid November 2003, we have implemented two Prolog meta interpreters for PeerTrust's run-time environment, each of them running in XSB Prolog as well as in MINERVA (a Java based Prolog engine). The meta interpreters evaluate guarded distributed logic programs as described in this paper, and have been tested on a set of examples exercising the different aspects of trust negotiation described in this paper. The meta interpreters, examples, and traces of their runs on all examples are available at http://www.learninglab.de/english/projects/peertrust.html. (The interested reader may find these examples and traces helpful in understanding exactly what transpires at run time when using the evaluation algorithm presented in the previous section.) The first meta interpreter simulates distributed evaluation and the knowledge bases of different peers by adding a *Peer* argument to each rule, and generates the proof trees one at a time, using the usual Prolog depth-first evaluation strategy. The second meta interpreter simulates distributed evaluation based on queues for the active proof trees for each peer, as described in the previous section.

We are currently extending the second meta interpreter to have each peer run on a separate host, which involves replacing the remote evaluation currently simulated within one Prolog process by socket based communication between two Prolog processes. The sockets must use a communication protocol that prevents eavesdropping, man-in-the-middle attacks, replay attacks, etc., in the manner of SSL/TLS and https. As the queues of each peer store all necessary evaluation context during computation, no further changes in the meta interpreter are needed to achieve true distributed evaluation. Furthermore, we are setting up the MINERVA policy evaluation engine as library that can be called from an arbitrary Java program, to directly use the trust negotiation facilities in ELENA testbeds.

We are also working on import/export routines from the Datalog-based QEL query language used in Edutella (see [19] and http://edutella.jxta.org/spec/qel.html for the language specification) to use PeerTrust's trust negotiation engine to provide access control in the Edutella network. Additionally, we are implementing import/export of policies from/to RuleML, using a RuleML parser and translator (RuleML to Prolog and back) from Carlos Damasio, in the context of the REWERSE Network of Excellence.

6. RELATED WORK

Issues associated with trust management in decentralized systems and privacy protection have received increasing attention in

recent years. For example, IBM's Trust Establishment (TE) Project [8] has developed a system for establishing trust according to policies that specify constraints on the contents of public-key certificates. They adopt the web-of-trust model, and have proposed two policy languages [8] based on XML. The TE system gathers supporting credentials as needed at run time, starting from issuer sites. Trust establishment is unilateral in TE.

IBM's *idemix* project [4] is representative of anonymous-credential-based approaches to trust establishment, which typically offer strong privacy guarantees. *idemix* uses zero-knowledge proofs to prove satisfaction of policies, so the exact contents of a credential are never revealed and the actions of an individual at different servers cannot be recognized as belonging to the same individual. It will be interesting to deploy *idemix*'s credentials in the more complex types of policies and n-party negotiations studied in this paper.

KeyNote is a capability-based trust management system from AT&T [2] that manages delegation of authority for a particular application. KeyNote is not for establishing trust between strangers, since clients are assumed to possess credentials that represent authorization of specific actions with the application server. Trust negotiation could be used to gain access to KeyNote credentials, so that a stranger could gain access to a resource protected by KeyNote.

The Secure Dynamically Distributed Datalog (SD3) trust management system [13] is closely related to PeerTrust. SD3 allows users to specify high level security policies through a policy language. The detailed policy evaluation and certificate verification is handled by SD3. Since the policy language in SD3 is an extension of Datalog, security policies are a set of assumptions and inference rules. SD3 can automatically contact a remote party to gather further credentials during the evaluation of a policy. SD3 literals include a "site" argument similar to our "Issuer" argument, though this argument cannot be nested, and "Requester" arguments are not possible either, which restricts SD3's expressiveness. SD3 does not consider the protection of sensitive credentials or policies, and does not have the notion of guards.

The P3P standard [27] focuses on negotiating the disclosure of a user's sensitive private information based on the privacy practices of the server. Trust negotiation generalizes this by basing resource disclosure on any properties of interest that can be represented in credentials. The work on trust negotiation focuses on certified properties of the credential holder, while P3P is based on data submitted by the client that are claims the client makes about itself. Support for both kinds of information in trust negotiation is warranted.

SSL, the predominant credential-exchange mechanism in use on the web, and its successor TLS support credential disclosure during client and server authentication. TLS is suited for identity-based credentials and requires extension to make it adaptable to trust negotiation [9]. Needed additions include protection for sensitive server credentials and a way for the client to explain its policies to the server.

Li et al. [16, 15] introduced the RT family of role-based trust-management languages, which can be used to map entities to roles based on the properties described in their credentials. RT has a Datalog-based semantics, and offers an elegantly uniform treatment of policies and credentials. The RT work has addressed issues related to delegation of authority, and PeerTrust's handling of delegation is very similar to RT's. The RT work has examined the problem of leaks during trust negotiation, and proposed *acknowledgement policies* as a way to prevent many undesirable inferences about which credentials a party possesses, the exact contents of

those credentials, and other information that can be inferred from a set of credentials and policies. PeerTrust implements acknowledgement policies for credentials as guards in Horn clauses. The RT work has also addressed the question of how to find relevant credentials at run time [16]. PeerTrust generalizes RT's approach by distributing the search process across the set of peers, and allowing a variety of credential search strategies to be encoded as declarative rules.

Yu et al. [28] have investigated issues relating to autonomy and privacy during trust negotiation. The work on autonomy focuses on allowing each party in a negotiation maximal freedom in choosing what to disclose, from among all possible safe disclosures. Their approach is to predefine a large set of negotiation *strategies*, each of which chooses the set of disclosures in a different way, and prove that each pair of strategies in the set has the property that if Alice and E-Learn independently pick any two strategies from the set, then their negotiation is guaranteed to establish trust if there is any safe sequence of disclosures that leads to the disclosure of the target resource. Then Alice and E-Learn only have to agree on which set of strategies they will use. Yu et al.'s approach to protecting sensitive information in policies is UniPro, which is supported by the run-time environment we have presented in this paper.

Recent work in the context of the Semantic Web has focussed on how to describe security requirements in this environment, leading to the KAoS and Rei policy languages described, for example, in [14] and [24]. KAoS and Rei investigate the use of ontologies for modeling speech acts, objects and access types necessary for specifying security policies on the Semantic Web. PeerTrust complements these approaches by targeting trust establishment between strangers and the dynamic exchange of credentials during an iterative *trust negotiation* process that can be declaratively expressed and implemented based on guarded distributed logic programs.

Similar to the situated courteous logic programs of [7, 6] that describe agent contracts and business rules, we build upon a logic programming foundation to declaratively represent policy rules and iterative trust establishment. The extensions described in [7, 6] are orthogonal to the ones described in this paper; an interesting addition to our guarded distributed logic programs would be the notion of prioritized rules to explicitly express preferences between different policy rules.

7. CONCLUSION AND FURTHER WORK

The Semantic Web poses formidable challenges for access control: no generally accepted centralized authorities, strong needs for local autonomy and privacy, a wide variety of resources requiring protection, and challenges in finding the information needed to show that a policy is satisfied. This paper has shown how to harness a network of semi-cooperative peers to automatically create, in a distributed fashion, a certified proof that a party is entitled to access a particular resource on the Semantic Web. We have also shown how to use a declarative policy and credential language to support crucial trust negotiation features such as delegation, bilateral iterative disclosure of credentials, and policy protection.

There are many compelling directions for future work on the use of distributed certified proofs as a basis for trust negotiation. Due to space limitations, we will single out only two of these directions. First, one would like to see formal guarantees that trust negotiations will always terminate and will succeed (i.e., result in access to the desired resource) when possible. Further, one would like to see an analysis of the autonomy available to each peer (e.g., "If I refuse to answer this query, could it cause the negotiation to fail?") and the information that can be leaked by a peer's behavior during negotiation. The first three kinds of guarantees are preordained for

all meta interpreters that implement the negotiation protocols and strategies proposed in [28], which are more complex than the simple meta interpreters presented in this paper, but offer peers a much higher degree of autonomy. Thus one interesting future direction is the extension of these strategies, which were designed for negotiations that involve exactly two peers, to work with the n peers that may take part in a negotiation under PeerTrust.

Second, the example policies in this paper each protect a single resource and a single type of access to that resource. For scalability, Semantic Web access control policies must support an intensional specification of the resources and types of access affected by a policy, e.g., as a query over the relevant resource attributes ("the ability to print color documents on all printers on the third floor"). This capability, already present in policy languages such as Rei, KAoS, and Ponder, is supported at run time by the *content-triggered* variety of trust negotiation [10]. We are currently working to adapt content-triggered trust negotiation to the context of the Semantic Web.

8. REFERENCES

[1] E. Bina, V. Jones, R. McCool, and M. Winslett. Secure Access to Data Over the Internet. In *Conference on Parallel and Distributed Information Systems*, Sept. 1994.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System Version 2. In *Internet Draft RFC 2704*, Sept. 1999.

[3] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. The MIT Press, 2000.

[4] J. Camenisch and E. Herreweghen. Design and Implementation of the *Idemix* Anonymous Credential System. In *ACM Conference on Computer and Communication Security*, Washington D.C., Nov. 2002.

[5] D. Chaum. Security without Identification: Transactions Systems to Make Big Brother Obsolete. *Communications of the ACM*, 24(2), 1985.

[6] B. Grosof. Representing e-business rules for the semantic web: Situated courteous logic programs in RuleML. In *Proceedings of the Workshop on Information Technologies and Systems (WITS)*, New Orleans, LA, USA, Dec. 2001.

[7] B. Grosof and T. Poon. SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *Proceedings of the 12th World Wide Web Conference*, Budapest, Hungary, May 2003.

[8] A. Herzberg, J. Mihaeli, Y. Mass, D. Naor, and Y. Ravid. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.

[9] A. Hess, J. Jacobson, H. Mills, R. Wamsley, K. Seamons, and B. Smith. Advanced Client/Server Authentication in TLS. In *Network and Distributed System Security Symposium*, San Diego, CA, Feb. 2002.

[10] A. Hess and K. E. Seamons. An Access Control Model for Dynamic Client Content. In *8th ACM Symposium on Access Control Models and Technologies*, Como, Italy, June 2003.

[11] I. Horrocks and P. Patel-Schneider. A proposal for an owl rules language. http://www.cs.man.ac.uk/ horrocks/DAML/Rules/, Oct. 2003.

[12] International Telecommunication Union. *Rec. X.509 - Information Technology - Open Systems Interconnection - The Directory: Authentication Framework*, Aug. 1997.

[13] T. Jim. SD3: A Trust Management System With Certified Evaluation. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.

[14] L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *Proceedings of the 2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.

[15] N. Li, J. Mitchell, and W. Winsborough. Design of a Role-based Trust-management Framework. In *IEEE Symposium on Security and Privacy*, Berkeley, California, May 2002.

[16] N. Li, W. Winsborough, and J. Mitchell. Distributed Credential Chain Discovery in Trust Management. *Journal of Computer Security*, 11(1), Feb. 2003.

[17] J. W. Lloyd. *Foundations of Logic Programming*. Springer, 2nd edition edition, 1987.

[18] W. Nejdl, W. Siberski, and M. Sintek. Design issues and challenges for RDF- and schema-based peer-to-peer systems. *SIGMOD Record*, 32(3), 2003.

[19] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmr, and T. Risch. Edutella: A P2P networking infrastructure based on RDF. In *Proceedings of the 11th International World Wide Web Conference (WWW2002)*, Hawaii, USA, June 2002.

[20] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser. Super-peer-based routing and clustering strategies for rdf-based peer-to-peer networks. In *Proceedings of the International World Wide Web Conference*, Budapest, Hungary, May 2003.

[21] B. Schneier. *Applied Cryptography, second edition*. John Wiley and Sons. Inc., 1996.

[22] 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.

[23] B. Simon, Z. Mikls, W. Nejdl, M. Sintek, and J. Salvachua. Smart space for learning: A mediation infrastructure for learning services. In *Proceedings of the Twelfth International Conference on World Wide Web*, Budapest, Hungary, May 2003.

[24] 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 *Proceedings of the 2nd International Semantic Web Conference*, Sanibel Island, Florida, USA, Oct. 2003.

[25] 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.

[26] K. Ueda. Guarded horn clauses. In *Logic Programming '85, Proceedings of the 4th Conference*, LNCS 221, pages 168–179, 1986.

[27] W3C, http://www.w3.org/TR/WD-P3P/Overview.html. *Platform for Privacy Preferences (P3P) Specification*.

[28] T. Yu, M. Winslett, and K. Seamons. Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies in Automated Trust Negotiation. *ACM Transactions on Information and System Security*, 6(1), Feb. 2003.

[29] P. Zimmerman. *PGP User's Guide*. MIT Press, 1994.