

PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web

Wolfgang Nejdl¹, Daniel Olmedilla¹, and Marianne Winslett²

¹ L3S and University of Hannover, Germany
{nejdl, olmedilla}@l3s.de

² Dept. of Computer Science, 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. 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 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 PeerTrust language based upon distributed logic programs, and compare it 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 in the ELENA distributed eLearning environment. Finally, we discuss related work and identify areas for further research.

Keywords: Automated Trust Negotiation, Peer-to-Peer, Semantic Web, 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 [9, 19], such as services, documents, roles, and capabilities. For example, in the Edutella infrastructure [15, 14, 16], 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 [18], 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 release 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 and information release on the Semantic Web [9, 19]: in order to see an appropriate subset of Alice's digital credentials, E-Learn will have to show that it satisfies the release 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 release 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 and release 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 and release 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 and information release. We start by introducing the concepts behind trust negotiation in section 2. We then introduce distributed logic programs to express and implement trust negotiation in a distributed environment, in section 3 and discuss PeerTrust's trust negotiation using distributed logic programs in detail in section 4. We discuss related work in section 5 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.

In contrast, 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 common.

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 and release 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 and release policies (*policies*, for short) 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 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, \dots, C_k, R) , where R is the resource to which access was originally requested, such that when credential C_i is disclosed, its 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 .)

In practice, trust negotiation is conducted by security agents who interact with each other on behalf of users. A user only needs to specify 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 policies and trust negotiation using 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 [17].

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 accredited).

Sensitive policies The information in a 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? 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 UniPro [21], which gives (opaque) names to policies and allows any named policy $P1$ to have its own 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 [11] and how to find credentials [12].

3 Distributed Logic Programs

3.1 Syntax

Definite Horn Clauses PeerTrust’s language is based on first order Horn rules (definite Horn clauses), i.e., rules of the form

$$lit_0 \leftarrow lit_1, \dots, lit_n$$

where each lit_i is a positive literal $P_j(t_1, \dots, t_n)$, 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 [13], which have been used as the basis for the rule layer of the Semantic Web and specified in the RuleML effort ([4, 5]) as well as in the recent OWL Rules Draft [7]. 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 instead 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 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.

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 *Authority* argument,

$$lit_i @ \text{Authority}$$

where *Authority* 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:
 $\text{preferred}(X) \leftarrow \text{student}(X) @ \text{“UIUC”}$.

This policy says that the UIUC peer is responsible for certifying the student status of a given person³. (For clarity, we prefix each rule by the peer in whose knowledge base it is included.)

The *Authority* argument can be a nested term containing a sequence of authorities, 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:
 $\text{student}(X) @ \text{“UIUC”} \leftarrow$
 $\text{student}(X) @ \text{“UIUC”} @ X$.

As mentioned earlier, CSP and UIUC may need a way of referring to the peer who asked a particular query. We accomplish this with *Context* literals that represent release policies for literals and rules, so that we now have literals and rules of the form

$$\begin{aligned} lit_i @ Authority \$ context_j \\ lit_i \leftarrow_{context_j} lit_1, \dots, lit_{i-1} \end{aligned}$$

For example, suppose that Alice has derived $\text{student}(\text{“Alice”}) @ \text{“UIUC”}$ and she wishes to send this literal to eOrg. She can only do so if she is able to derive $\text{student}(\text{“Alice”}) @ \text{“UIUC”} \$ \text{Requester} = \text{“eOrg”}$. Here, *Requester* is a pseudovalue whose value is automatically set to the party that Alice is trying to send the literal or rule. If no context is specified for a literal or a rule, the default context ‘Requester = Self’ applies, implying that the literal or rule cannot be sent to any other peer. ‘Self’ is a pseudovalue whose value is a distinguished name of the local peer. The release policy for a literal can be cleanly specified in rules separate from those used to derive the literal, e.g.,

$$p(X_1, \dots, X_n) \$ context_p(X_1, \dots, X_n, \text{Requester}, \text{Self}) \leftarrow p(X_1, \dots, X_n)$$

In this paper, we will strip the contexts from literals and rules when they are sent to another peer. However, sticky policies can be implemented by leaving contexts attached to literals and rules in messages and defining how to propagate contexts across modus ponens, so that a peer can control further dissemination of its released information in a non-adversarial environment.

³ In practice, this policy must be written as $\text{preferred}(X) \leftarrow \text{student}(Y) @ \text{“UIUC”}$, $\text{authenticatesTo}(X, Y)$, 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.

Using the *Authority* and *Context* 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:

```
E-Learn:
freeEnroll(Course, Requester) $ true ←
  policeOfficer(Requester) @ "CSP" @ Requester,
  spanishCourse(Course).
```

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 runtime system may choose to include it directly in a message sent during the negotiation, as discussed later.

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 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 signed 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 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. For example, the UIUC registrar can use a signed rule to prove that it is entitled to determine who is a student at UIUC:

```
“UIUC Registrar”:  
student(X) @ “UIUC” ←  
  signedBy [“UIUC”]  
student(X) @ “UIUC Registrar”.
```

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.

3.2 Semantics

The semantics of the PeerTrust language is an extension of that of SD3 [20]. For each Authority argument that has not been specified explicitly in a rule or literal, we add the argument ‘@ Self’. We also add the notion of distributed *Peers* with their respective knowledge bases. The meaning of a PeerTrust program is determined by a forward chaining nondeterministic fixpoint computation process in which at each step, a non-deterministically chosen peer either applies one of its rules, sends a literal or rule in its knowledge base with context ‘Requester = P’ to peer P (after removing its context *and* digitally signing it), or receives a context-free signed rule or literal from another party. Axioms not shown here allow peers to convert signed literals ‘lit [signedBy A]’ to unsigned literals ‘lit @ A’ that can be used in applications of rules, and to strip signatures off rules as well. Entailment rules not shown here allow peers to apply modus ponens using rules and literals of their own (‘@ Self’) or that they have obtained from other peers (‘@ Alice’), so that they can mimic the reasoning processes of other peers. Due to space constraints, we omit all details in this paper. We also omit a discussion of query syntax; queries are needed for PeerTrust sites that use backward chaining.

In PeerTrust, many true statements will be underivable at a particular peer at runtime, because peers will not be willing to devote unlimited time and effort to trying to answer the queries of other peers. Each peer can control how much effort it is willing to exert to help other peers; most peers will only be willing to answer a few kinds of queries, and those only for a few kinds of requesters. In this paper, we do not present possible evaluation schemes for peers, but obvious choices include a forward-chaining ‘push’ paradigm and a backward-chaining paradigm based on an extension of SLD resolution. Also, in practice, it is not necessary for the contents of every message to be signed (e.g., there is no real need for Alice to sign her UIUC student ID before sending it to eOrg), but space constraints impel us to omit a discussion of the syntax and semantics that can be used to avoid message signatures in some cases.

4 Automated Trust Negotiation in Detail: Examples and DLPs

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 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:

E-Learn:

```
discountEnroll(Course, Party) $ Requester = Party ←
  discountEnroll(Course, Party).
discountEnroll(Course, Party) ←
  eligibleForDiscount(Party, 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 UIUC’s release policies are unlikely to allow release of student information to E-Learn. So instead E-Learn will ask students themselves to provide full proof of their student status, as hinted at by the following rule:

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

This rule follows directly from PeerTrust axioms, but its inclusion directly in E-Learn’s program is a hint to E-Learn’s runtime evaluation engine (not shown here) that the engine should not try to evaluate the rule itself. In the current implementation of PeerTrust, each ‘@ authority’ argument is taken as a directive to the runtime engine regarding who should try to evaluate that particular literal.

E-Learn is a member of the Better Business Bureau, and can prove it through an appropriate release policy (not shown) and signed rule:

```
member(“E-Learn”) @ “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.

UIUC:

```
student(X) $ Requester = “UIUC Registrar” ←
  student(X) @ “UIUC Registrar”.
```

UIUC does not directly respond to queries about student status, or release its delegation rule to anyone other than the registrar. Students get a credential from the UIUC registrar certifying their student status, and a copy of the delegation rule that UIUC gave to the UIUC registrar. Alice has both of these. Her policy is to give out her credentials only to members of the Better Business Bureau, and she expects them to produce a proof of this membership themselves, as hinted at by her inclusion of multiple levels of required signatures in her (publicly releasable) release policy for student literals:

Alice:

```
student(X) @ Y $ member(Requester) @ "BBB" @ Requester ←true
  student(X) @ Y
```

With the current implementation of the PeerTrust 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. 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") @ X $ member(Requester) @ "ELENA" ←true
  employee("Bob") @ X.
employee("Bob") @ "IBM" ←
  signedBy ["IBM"].
authorized("Bob", Price) @ X $ member(Requester) @ "ELENA" ←true
  authorized("Bob", Price) @ X.
authorized("Bob", Price) @ "IBM" ←
  signedBy["IBM"]
  Price < 2000.
member(Requester) @ "ELENA" ←true
  member(Requester) @ "ELENA" @ Requester.
```

Bob pays with his company's credit card, but will not even discuss the existence of the card with non-members of ELENA. 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 we show only a name field, containing "IBM". From previous interactions, Bob also knows that IBM and E-Learn are members of the ELENA consortium.

```
visaCard("IBM")
  signedBy ["VISA"].
visaCard("IBM") $ policy27(Requester) ←true
```

```

visaCard("IBM").
policy27(Requester) ←
  authorizedMerchant(Requester) @ "VISA" @ Requester,
  member(Requester) @ "ELENA".
member("IBM") @ "ELENA"
  signedBy ["ELENA"].
member("E-Learn") @ "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; the "extra" variables in some rule heads are needed by those external functions.

```

"E-Learn":
enroll(Course, Requester, Company, Email, 0) ←true
  freeCourse(Course),
  freebieEligible(Course, Requester, Company, EMail).
enroll(Course, Requester, Company, Email, Price) ←true
  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) ←true
  price(Course, Price),
  authorized(Requester, Price) @ Company @ Requester,
  visaCard(Company) @ "VISA" @ Requester.

```

The use of policy names and contexts 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, the definition of freebieEligible has the default context (Requester = Self). ELENA member companies can disseminate the definition 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.

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("E-Learn")
  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 interpretation for the revocation speech acts mentioned in [9].) 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) ←true
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 *Authority* 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) ←true
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:

```
policy49(Course, Requester, Company, Price) ←true
price(Course, Price),
authorized(Requester, Price) @ Company @ Requester,
visaCard(Company) @ "VISA" @ Requester,
authority(purchaseApproved, Authority) @ myBroker,
purchaseApproved(Company, Price) @ Authority.
```

With the PeerTrust 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 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 Related Work

The Secure Dynamically Distributed Datalog (SD3) trust management system [8] 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 literals include a "site" argument similar to our "Authority" argument, though this argument cannot be nested. SD3 does not have the concept of a context, which is appropriate for SD3's target application of DNS, but restricts SD3's expressiveness too much for our purposes. SD3 does not have a mechanism to allow the information in policies to be kept private from certain parties, which we accomplish with contexts. The newly proposed policy language Cassandra [2] combines many of the features of SD3 and RT [10], and is also close to PeerTrust.

Yu et al. [21] 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. Similar concepts will be needed in PeerTrust. 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, leading to the KAoS and Rei policy languages [9, 19]. 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 certificates during an iterative trust negotiation process that can be declaratively expressed and implemented based on distributed logic programs.

Similar to the situated courteous logic programs of [5] that describe agent contracts and business rules, PeerTrust builds upon a logic programming foundation to declaratively represent policy rules and iterative trust establishment. The extensions described in [5] are orthogonal to the ones described in this paper; an interesting addition

to PeerTrust's distributed logic programs would be the notion of prioritized rules to explicitly express preferences between different policy rules.

6 Conclusion and Further Work

In this paper, we have used discussed the PeerTrust policy language and how to use it for negotiating and establishing trust in a distributed elearning environment investigated in the EU/IST ELENA project (another scenario of using PeerTrust in a Grid environment is described in [1]). PeerTrust harnesses 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.

For readers interested in experimenting with PeerTrust, the PeerTrust 1.0 prototype is freely available at <http://www.learninglab.de/english/projects/peertrust.html> and <https://sourceforge.net/projects/peertrust/>. Like the earlier prototype described in [3], PeerTrust 1.0's outer layer is a signed Java application or applet program, which keeps queues of propositions that are in the process of being proved, parses incoming queries, translates them to the PeerTrust language, and passes them to the inner layer. Its inner layer answers queries by reasoning about PeerTrust policy rules and certificates using Prolog metainterpreters (in MINERVA Prolog, whose Java implementation offers excellent portability), and returns the answers to the outer layer. PeerTrust 1.0 imports RDF metadata to represent policies for access to resources, and uses X.509 certificates and the Java Cryptography Architecture for signatures. It employs secure socket connections between negotiating parties, and its facilities for communication and access to security related libraries are in Java. PeerTrust 1.0 implements an earlier version of the policy language presented in this paper; in particular, contexts are just simple Requester arguments, rather than arbitrary predicates; and DLPs are used to provide 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 [21], 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 [6]. We are currently working to adapt content-triggered trust negotiation to the context of the Semantic Web.

Acknowledgments The research of Nejdl and Olmedilla was partially supported by the projects ELENA (<http://www.elena-project.org>, IST-2001-37264) and REVERSE (<http://reverse.net>, IST-506779). The research of Winslett was supported by DARPA (N66001-01-1-8908), the National Science Foundation (CCR-0325951, IIS-0331707) and The Regents of the University of California.

References

1. J. Basney, W. Nejdl, D. Olmedilla, V. Welch, and M. Winslett. Negotiating trust on the grid. In *2nd SemPGRID Workshop*, New York, USA, May 2004. co-located with WWW’2004.
2. M. Y. Becker and P. Sewell. Cassandra: distributed access control policies with tunable expressiveness. In *Policies in Distributed Systems and Networks*, June 2004.
3. R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *European Semantic Web Symposium*, Heraklion, Greece, May 2004.
4. B. Grosz. 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.
5. B. Grosz and T. Poon. SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In *WWW12*, 2003.
6. 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.
7. I. Horrocks and P. Patel-Schneider. A proposal for an owl rules language. <http://www.cs.man.ac.uk/~horrocks/DAML/Rules/>, Oct. 2003.
8. T. Jim. SD3: A Trust Management System With Certified Evaluation. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.
9. L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *International Semantic Web Conference*, Sanibel Island, Oct. 2003.
10. N. Li and J. Mitchell. RT: A Role-based Trust-management Framework. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, Washington, D.C., Apr. 2003.
11. 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.
12. N. Li, W. Winsborough, and J. Mitchell. Distributed Credential Chain Discovery in Trust Management. *Journal of Computer Security*, 11(1), Feb. 2003.
13. J. W. Lloyd. *Foundations of Logic Programming*. Springer, 2nd edition edition, 1987.
14. 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.
15. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, 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.

16. 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.
17. 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.
18. B. Simon, Z. Miklós, 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.
19. 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 International Semantic Web Conference*, Sanibel Island, Oct. 2003.
20. J. Trevor and D. Suciu. Dynamically distributed query evaluation. In *PODS*, 2001.
21. 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.