

# Policy-driven Negotiation for Authorization in the Semantic Grid

Ionut Constandache<sup>a</sup> Daniel Olmedilla<sup>a</sup> Frank Siebenlist<sup>b</sup> Wolfgang Nejdl<sup>a</sup>

<sup>a</sup> *L3S Research Center and University of Hanover, Germany*

*{constandache,olmedilla,nejdl}@l3s.de*

<sup>b</sup> *Argonne National Laboratory, USA*  
*franks@mcs.anl.gov*

---

## Abstract

As in many Grid Services deployments the clients and servers reside in different administrative domains, there is both a requirement to discover each other's authorization policy in order to be able to present the right assertions that allow access, as well as to reveal as little as possible of the access policy details to unauthorized parties. This paper describes a mechanism where the client and servers are semantically annotated with policies that protect their resources. These annotations specify both constraints and capabilities, which are used during a negotiation to reason about and to communicate the need to see certain credentials from the other party, and to determine whether requested credentials can be obtained and revealed. The end result of the negotiation is a state where either both parties have satisfied their policy constraints for a subsequent interaction, or where such interaction is disallowed by either or both. Furthermore, the implementation of a prototype is discussed that is based on the PEERTRUST policy language and a reasoning engine, which are integrated in the webservices runtime of the Globus Toolkit. The negotiation process is facilitated through the implementation of WSRF-compliant service interfaces for the protocol message exchanges.

*Key words:* Grid, Web Services, Authentication, Authorization, policy, negotiation, Globus Toolkit, Semantic Web, WSRF, PEERTRUST

---

## 1. Introduction

Organizations join in collaborations for the benefit of sharing resources for data retrieval, job execution, monitoring, and data storage. Each organization is defined by its own administrative domain, while the overlaying Virtual Organization (VO) is defined by the collaboration agreement. Grid toolkits provide the middleware for the applications such that the shared resources residing in the different domains can be securely accessed. Such an environment provides users with seamless access to all resources they are authorized to. In current Grid infrastructures, in order to

be granted access at each domain, user's jobs have to secure and provide appropriate digital credentials for authentication and authorization. However, while authentication along with single sign-on can be provided based on client delegation of X.509 proxy certificates to the job being submitted, most authorization mechanisms are still identity based. Due to the potentially large number of users and different certification authorities, this leads to scalability problems, which require alternative solutions.

Semantic Web technologies allow for the annotation of grid services in a machine-understandable language. We exploit this feature in the context of authentication and authorization by binding rule-based poli-

cies to resources. These policies specify access control requirements that must be satisfied by a requester before it is authorized. In addition, we claim that authorization in the Semantic Web cannot be based on parties' identities alone as it does not scale, and hence is not anymore a one-shot mechanism. Instead, it is an iterative process in which the entities involved must be able to negotiate and incrementally increase their level of trust based on other party's properties. The application of Semantic Web authorization mechanisms to Grid environments overcome current Grid authorization limitations. Policy-based negotiations provide scalability, advanced access control and automatic credential fetching. These ingredients enable a complete set of new scenarios in the context of authorization in which user involvement is drastically reduced (from administrative and development points of view) in favor of automated interactions.

This paper identifies current limitations in Grid authorization, suggests the application of policy languages to protect access to resources and demonstrates the power and advantages of this approach. In addition, we propose an architecture in which policy languages can be easily integrated into the Globus Toolkit 4.0 and allowing advanced access control and automatic credential fetching. Finally we describe how this architecture has been implemented and tested using the Semantic Policy Language PEERTRUST for policy specification.

## 2. Grid-specific Authentication and Authorization

The Globus Toolkit (GT) [1] offers a collection of software that implements different protocols, specifications, standards and interfaces to meet the requirements identified by the Open Grid Services Architecture (OGSA) [2] to support a Grid Infrastructure. Ever since its initial inception in the late 1990s, the Globus Toolkit has addressed issues like: resource discovery, remote execution monitoring and management, data movement and replication, and security in service oriented distributed environments. In the following subsections we provide a short introduction to the Globus Toolkit 4.0 (GT4.0) [3] focusing on its integrated Grid Security Infrastructure and discuss some of the commonly used credential repositories used in Grid de-

ployments.

The requirement for secure communication between entities on the Grid has motivated the development of the Grid Security Infrastructure (GSI) [4]. GSI provides integrity protection, confidentiality and authentication for sensitive information passed over the network as well as the facilities to securely traverse the different organizations that are part of a collaboration. In this section we will summarize the characteristics of the GSI currently supported in GT4.0.

The fundamental security mechanism used by the GSI infrastructure is based on public key cryptography and the associated PKIX X.509-based Public Key Infrastructure (PKI) [5]. In a PKI, a unique key pair is associated with each party and entity. Data can be encrypted using one key, and decrypted with the other. Entities make one of the keys publicly available, which becomes the so-called public key. The other key is securely stored and not shared by the entity, and is known as the private key. By using the private key, the holder can encrypt messages and send them over the network. Such messages can be decrypted only with the associated public key. If the counterpart can decrypt a message using the public key of an entity, it is assured that no other entity could have sent that message as the associated private key is assumed to be securely kept by the entity holding it. Being able to decrypt the message with the public key of a certain entity, authenticates to the receiver that party as being the source of the message.

A PKIX compliant PKI uses X.509 identity-certificates to bind a public key to a unique name identifying the private key holder (see figure 1). Certificates contain the following information: a distinguished name (DN) which uniquely identifies the entity holding the certificate, the public key belonging to the DN identified in the certificate, the DN of the Certification Authority (CA) signing the certificate and thus attesting the relationship between the DN and the public key of the certificate, the CA digital signature and an expiration date. It is important to observe that a trusted CA is used to certify the DN - public key binding in the certificate. The CA guarantees through its signing policy that the two belong together and attests to this by signing the certificate.

In the GSI, all entities authenticate themselves through X.509 End Entity Certificates that are issued by a CA. The authentication protocol requires that

each party trusts the other party's CA, and is able to prove the ownership of the private key associated with the public key of its certificate.

The private key of a certificate is usually stored encrypted with a passphrase, in a file residing on the local file system. This is done to prevent the use of the private key in case the file containing it is compromised. Note that any unauthorized entity who has access to the private key, can impersonate the rightful holder of the associated certificate.

As we have seen, the authentication process involves signing through the use of the private key. Each time authentication is required the user has to decrypt his private key by entering the protecting passphrase. In a Grid environment, where the user's programs often access many different resources, typing this passphrase for each access is not very practical, especially when resources are accessed at unpredictable times.

To address this issue, the GSI make use of a delegation mechanism through the use of X.509 Proxy Certificates [6]. An X.509 Proxy Certificate is a short lived certificate that is generated together with an associated private key by the user, and signed through the user's X.509 End Entity Certificate credential. As a consequence, if a proxy certificate private key is compromised, then because of the proxy-certificate's short lifetime (several hours) the harm that can be inferred is limited. Therefore, the proxy certificate private key is normally only protected through the local file system access permissions, which allows applications to use the proxy certificate's private key without any additional user input. If the proxy certificate expires, the user can generate a new proxy certificate. Note that it is possible to automatize this process such that long running jobs can work without user intervention.

Proxy certificates are also used to delegate the user's rights to agents, intermediaries or resources such that they can interact with other resources on the user's behalf. The issuing process is identical to the one where the user generates a proxy certificate for himself, but in this case the agent will generate a new key-pair and the user will sign and issue a proxy certificate that holds the agent's public key, and in effect empower that agent with the user's rights. A proxy certificate can be signed by an end entity certificate or by another proxy certificate. The latter allows an agent holding a delegated certificate to delegate the associated user's

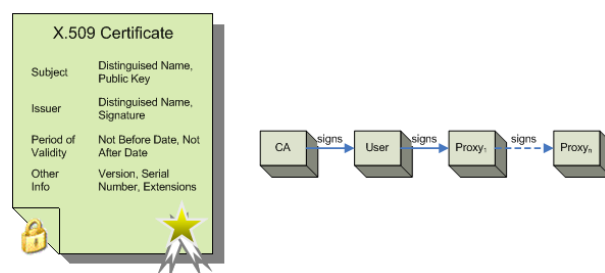


Fig. 1. X.509 Certificate and Proxy Chain Certificate Validation

rights to other agents that have to work on the user's behalf. During the authentication process, the whole chain of (proxy-)certificates is exchanged between the parties, which allows each relying party to validate the issued certificates and to obtain the identity information of the user on who's behalf the request is made from the end entity certificate at the start of the chain (see figure 1) The user's identity information is authenticated if that user's end entity certificate is signed by a trusted CA.

GT4.0 supports an Authorization Framework [7] for enforcing authorization on both client and service side. On the service side, authorization mechanisms rely on a configured chain of Policy Decision Points (PDPs) to determine if authorization should be granted or denied for a client invocation. The framework allows the plugin of custom PDPs. The decision regarding authorization is made based on the conjunction of all PDPs decisions, that is, authorization is granted only if all PDPs have returned a permit decision. The PDP's decision logic can be implemented based on the client's DN, the resource accessed and the operation invoked.

Services and clients configure security options either programmatically or through security descriptors. Security descriptors are represented in XML format, conform to a defined schema, and specify the certificates and the methods to be used for message protection, authentication and authorization.

On the service side, a number of PDP implementations for different authorization mechanisms are part of the GT4.0 distribution. Through security descriptor settings, these PDPs are either configured at service level or for the entire GT4.0 container running the service. The available options are: `self` (identity of the service and client are expected to be the same), `gridmap` (the identity of the client must be mapped to a local user account on the resource in a gridmap

file), `identity` (a certain identity of the client is expected where `identity` refers to the DN present in client certificate), `host` (a certain host name is expected of the entity requiring access), `samlCallout` (a SAML authorization callout to an external OGSA Authorization compliant service [8]), `xacml` (an embedded XACML [9] authorization engine evaluates the access decision), `userName` (username and password based authorization). On the client side, service authorization options are: `self`, `host` or `identity`.

Credential repositories offer centralized, persistent storage of user credentials, and allow for dynamic retrieval of those credentials by either the user himself or by entities delegated to act on the user's behalf. The repositories are considered sensitive infrastructure services and are expected to be hosted in data centers that provide adequate protection.

The secure maintenance of the long-term private key store for the end entity credentials is often a challenge for the user. It has become almost impossible to protect the user's machines from viruses and trojans, which in turn could compromise the user's private key. Furthermore, many users deploy multiple desktops from which they launch Grid applications, and therefore require a copy of their credentials on all those different systems they work from, with the inherent security risks of replication of secrets. For all those reasons, it is often easier to protect the user's private keys in a secure centralized credential repository.

The *MyProxy Online Credential Repository* [10] addresses these issues by providing a repository for secure credential storage and retrieval over the network. MyProxy Repositories allow storage of client credentials in the form of X.509 End Entity Certificates or X.509 Proxy Certificates and support the subsequent retrieval of delegated X.509 Proxy Certificates with a short lifetime. The protocol used by the MyProxy Repository is the same as used for the GSI delegation capability where the delegated key-pair is generated on the user's machine and the proxy certificate is signed on the MyProxy server by the private key of a user's certificate stored with the repository. Private keys are not transferred over the network thus minimizing risk of theft. Clients have user accounts and passwords set with the MyProxy Credential Repository and can store credentials and encrypt the private keys with their own supplied passwords. Only if a correct password is provided the private key of the delegating credential on

the MyProxy Repository can be decrypted and a valid X.509 Proxy Certificate be issued.

In many Grid deployments user job submission is handled through Grid Portals (web interfaces to the Grid environment). Such Grid Portals need delegated credentials for submitting jobs on user behalf. By providing the Grid Portal with the password protecting a previously stored credential on a MyProxy Server, the portal can retrieve a delegated X.509 Proxy Certificate and use it for client job submission.

MyProxy supports also the renewal of user credentials for long running jobs. A scheduler (e.g., Condor-G [11]) can monitor the job's credentials and, close to expiration, request a proxy certificate renewal from a MyProxy Credential Repository. In this situation, if the DN of the service running the scheduler matches the user supplied trusted services DNs and the service also proofs the possession of a previous delegated credential, the job credential is renewed.

A different approach that does not directly deal with client credentials is the *Community Authorization Service (CAS)* [12]. It permits definition and retrieval of authorization policies managed at the Virtual Organization (VO) [13] level. Collaborating organizations often form a VO and establish associated policies that govern the access rights over the shared resources. CAS provides a common point for the VO members to establish, administer, maintain consistent policies, and to release these policies to collaborating sites. Usually a CAS server is installed per VO with an identity known by all VO members and resource providers. The resource providers essentially empower the CAS through their local resource policy to administer the access rights within the VO. A user interested in accessing a resource, retrieves from the CAS server a SAML [14] signed assertion containing the rights of the user as allowed by the VO. The assertion contains the identity of the user and information about what actions are permitted on what resource, and is signed by the CAS identity. The user embeds the assertion in a proxy certificate and presents it to the resource with the service request. At the resource, the assertion is cryptographically validated. After that, the resource ensures that the assertion issuer, i.e. CAS, is allowed to administer the access rights. Lastly, it enforces the policy stated in the assertion as issued by CAS as it applies to the requester.

## 2.1. Example Scenario: Interaction between Jobs and Services in GT4.0

Let us consider the following illustrative scenario depicted in figure 2. A group of scientists at the Computer Science and Engineering Department of University “Politehnica” of Bucharest (UPB) would like to obtain data regarding oceanic water waves in order to develop signaling instruments for tsunami hazards avoidance. Fortunately, the university and the Navy Institute have an agreement, which allows the UPB members to use any of the Institute scientific instruments as long as they are not already in use, and as long as UPB has not exceeded its maximum number of monthly allocated hours for the use of institute’s resources. Both the university and the Navy Institute support the Globus Toolkit 4.0, which provides the common interoperable middleware-layer the scientists need to collect the desired data from a Wave Tank available at the institute site.

Alice, the leader of the group of scientists, has delegated to the UPB MyProxy Repository a X.509 Proxy Certificate derived from her UPB CA issued X.509 End Entity Certificate. Using this certificate the repository can sign other proxy certificates and deliver them to entities that have to act on Alice’s behalf. A requester can retrieve a proxy certificate if he can provide the username and password protecting the delegated credential on the repository. Also a requester can renew a proxy certificate if he can prove the possession of a previous delegated credential and if his own DN matches a regular expression setup for that particular proxy certificate on the MyProxy Repository.

Alice prepares a job and submits it over the Grid Infrastructure. First, she logs into the UPB Grid Portal and instructs the portal to retrieve, on her behalf, a delegated X.509 Proxy Certificate from the UPB MyProxy Repository, which will be used for the job. The Grid Portal authenticates using its own credential to the MyProxy Repository and since the DN of the certificate presented is in the allowed retrieve list of the repository, permission is granted and the request is made for a delegated certificate. Together with the request for the proxy certificate, the Grid Portal also sends the username and password provided by Alice. After those are verified by the MyProxy Repository, a proxy certificate for the job to the portal is issued on

Alice’s behalf. As a result, Alice’s job ends up with a chain of certificates containing Alice’s X.509 End Entity Certificate, the X.509 Proxy Certificate previously delegated to the MyProxy Repository and the X.509 Proxy Certificate delegated to the job on the portal. This chain of certificates identifies the job as pertaining to Alice.

The job is sent to the UPB HPC Center Linux cluster where it will retrieve the input data needed to setup the Wave Tank, perform cpu-intensive computations to refine and error correct the output data of the Wave Tank experiment, and store the results. First, the Linux cluster has to authenticate and authorize the submitted job. The certificate chain provided with the job has at its root Alice’s X.509 End Entity Certificate signed by the UPB Certification Authority(CA). The CA is trusted at the HPC Center Linux Cluster thus the job gets authenticated as running on Alice’s behalf. Authorization is also granted as a grid map-file entry maps Alice’s DN to a local user account permitting the job to start its work with the privileges of the associated local user.

As the job needs additional resources a new X509 Proxy Certificate is delegated by the Grid Portal to the UPB HPC Center Linux Cluster. Using this additional proxy certificate the job can authenticate and gain access to other Grid resources. First it will contact the UPB Reliable File Transfer Service (RFT) [15] to retrieve the input data. Authentication and authorization are satisfied based on Alice’s DN in the similar fashion as described before. Note that the chain of certificates presented to the RFT is one proxy certificate longer as it also contains the proxy certificate just delegated to the Linux Cluster.

Alice has already been informed that the Navy Institute requires a proof of her University affiliation, so she has instructed the job to contact the UPB CAS Server and retrieve a statement attesting her involvement with the university. After the required authentication and identity based authorization, the job obtains a SAML assertion attesting Alice university affiliation. The job processing application subsequently creates a new X.509 Proxy Certificate embedding the previous obtained assertion. This proxy certificate is going to be used to gain access at the Navy Institute Wave Tank site.

The Wave Tank forwards the authorization decision as a SAML authorization callout to the Navy Institute

Authorization Service, which verifies the assertion in the provided certificate. Assuming that all the other wave tank local requirements are fulfilled (the wave tank is not in use and UPB has not exceeded its allocated hours) data starts to arrive at the UPB HPC Center Linux Cluster.

Unfortunately the task proves to be a long one and the credentials delegated to the Linux Cluster are about to expire. The Linux Cluster has to renew its credential, so it contacts the UPB MyProxy Repository requesting a renewed credential. The distinguish name in UPB HPC Center Linux Cluster own certificate is in the renew allowed list of the MyProxy Repository and matches Alice's specified regular expression for requesters that are allowed to renew proxy certificates derived from her end entity certificate. The Linux Cluster is also able to prove the possession of a previous delegated credential. As a result, the MyProxy service issues a renewed credential to the Linux Cluster on Alice behalf.

Finally when the data has been corrected and refined the Linux Cluster contacts the UPB RFT Service, and after another round of authentication and authorization verification, the final data is made available to Alice and her group at the university file server.

## 2.2. *Current Limitations and Assumptions*

As we have observed during the interactions depicted in the previous scenario, authorization decisions were made based on user identity. The user had already an account set-up at the remote locations with a grid map file entry mapping his X.509 certificate's Subject's DN to his local account. Identities were already known by the MyProxy Server as it checked the requester's DN against the allowed retrieve and renew lists. Managing identity based access control lists requires site administrators to keep track of all possible clients allowed to request services. This is a difficult task as the user mapping entries are the result of the user's associations with certain organizations and projects. Once such relations cease to exist, the resource administrators have to remove all these entries to ensure that access is no longer allowed. On the other hand as soon as new users are required to deploy the Grid infrastructure, new mapping entries have to be entered in all relevant access lists to accommodate

the new identities. In our example we assumed that the university administrators have carefully set up the available resources, so that the university students and staff may put them to the best use. This may indeed not be an issue for small groups associated with a single institution but for large scale Grid deployments, each of the computational resources used by a job may be under a different authority. Entities with no previous interactions may have to communicate and it is unrealistic to assume that in large scale Grids, identity based authorization is a practical solution in such an environment.

Another assumption in the above scenario was that the UPB CA is trusted at each location the job needs access (UPB resources and Navy Institute Wave Tank). There has been no intermediary step for the job and the contacted resources to argue about trusted authorities. The trusted CAs of each site are expected to be known by the client who is also expected to provide an appropriate certificate.

In these conditions, mapping client identities to local accounts raises serious scalability problems due to the large number of potential users. Even more unrealistic is the requirement of having a single trusted Certification Authority when the Grid spans multiple organizations each having their own authentication infrastructure.

The use of attribute-based certificates is starting to become more popular as is shown by technologies like PRIMA [16], VOMS [17], CAS [12] and X.509 attribute certificates [18]. Their use allows clients to be mapped to local accounts based on their attributes. However, there is no standard interface to use these certificates as of yet, and their integration is not common practice in the current Grid deployments.

In all the previously described interactions, the authorization decision was a one step process, without the client asking about the resource access requirements and without any negotiation for the granting of authorization. The client was simply expected to be aware of and be able to satisfy the service requirements. As shown in section 2, the GT 4.0 options for client authorization are: none, username, host, identity, grid map-file, xacml or a callout to a SAML compliant authorization service, while service authorization options are: none, self, host or identity. Each of these options consists of a single interaction resulting in either access granted or refused. There is no avail-

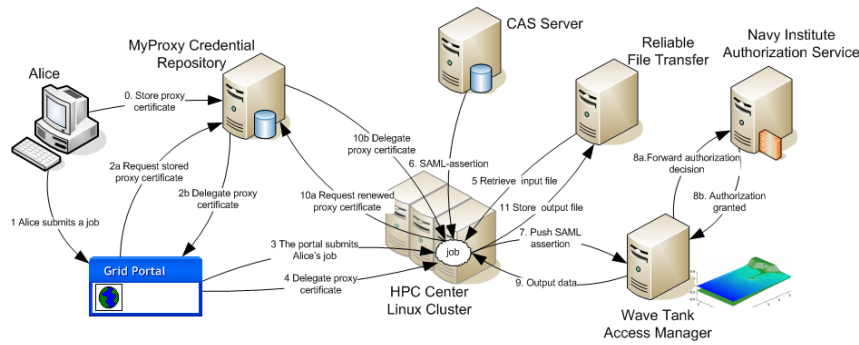


Fig. 2. Simple Grid Scenario

able API for client queries or service advertisement of authorization requirements, through which a client could ensure that it has all required credentials before making the actual request. Neither can the service dynamically inform the client about the locally trusted third party credential issuers.

When we consider an example where a job spawns a large number of sub-jobs and each of those sub-jobs has to find the best resources available on the Grid to be executed, then in such a setting the sub-jobs have to find a way to inform themselves autonomously of the authentication and authorization requirements at each identified resource and find automatically whether or not they can meet these requirements.

Some of the deployed Grids base their authorization on assertions issued by infrastructure services established at the virtual organization level (e.g., CAS). Grid service providers may feel uncomfortable to rely only on remote assertions especially in the case of a larger Grid environment. This could result in a preference to use, publish and enforce policies in the local domain such that the administrators keep fine grained access control over the resource they provide. Note that even if authorization statements are made by third parties at the VO level, the authorization decision may involve local information and state, as was shown in our previous example where the policy stated that the wave tank should not be in use and the University use of instruments should not have exceeded a certain limit before access was granted.

The client might have his own concerns regarding the resources accessed and the credentials he has to provide. He may ask the service itself to provide additional credentials before a trust relationship can be es-

tablished, or he may protect his credentials through a set of policies such that only after the service has satisfied those policy constraints, the protected credentials are revealed.

As Grid projects can share many resources that are located in multiple administrative domains, it will be an additional challenge to keep track of all the different credentials in relation to the resource and the domain where they are needed. This holds true also for the submitted job due to the difficulty in predicting the job's needs prior to its execution. When the job is expected to present a required and missing credential, the user may not be available. The alternative of providing the job with all the user's credentials might imply disclosing sensitive information with which the user may not be comfortable with. The issue for the user/job is to provide just the right/required credentials to the Grid service it tries to access - no more, no less.

We argue in the following sections that when Grid deployments support the specification, advertisement and enforcement of service level access control policies, together with capabilities for the automatic fetching of credentials, large scale collection of resources can indeed be supported. It will also enable the dynamic negotiation for authorization and access granting based on the properties of both parties.

### 3. Semantic Web and Rule-based Policy Languages

The term *policy*, in its more general sense, can be defined as "a statement describing the behavior of a

system". In terms of security and trust management, a policy defines which are the conditions a requester must satisfy in order to be granted access to a resource. Rule-based policy languages have been extensively used in the context of security policies, trust management and business rules [19]. The reason why researchers have typically selected rule-based languages to express policies is not arbitrary. Declarative rules are easier and faster to write than imperative code. Their semantics is closer to the way humans think and specially useful for access control protection where one specifies *what* conditions are to be fulfilled by the requester, without specifying *how*. Furthermore, rule-based policies provide self-described statements which can be exchanged, shared and reused among parties, and which allows one to reason over them, hence enabling interoperability and contributing to the Semantic Web vision. In fact, definite Horn clauses are the basis for logic programs, which have been used as the basis for the rule layer of the Semantic Web and specified in the RuleML effort [20]. In addition, it is already proposed to use policy reasoning in different scenarios of the Semantic Web like, for instance, Semantic Web Services [21].

In the following two subsections we present first how trust establishment might be enhance with policy-based negotiations followed by a description of the policy language that we will use in the rest of the paper.

### 3.1. Policy Based Negotiation for Trust Establishment

Security in distributed environments like the World Wide Web, Peer-to-Peer (P2P) systems or Grids, were traditionally built under the assumption that service providers and consumers are known to each other. In common scenarios, before allowing access to (possibly) sensitive resources, trust relations are established among entities by having clients registered with the resource they try to access. This process may involve the creation of an account, profile, and the addition of the user identity to some kind of access control lists, or some offline contacts specifying exactly who and what kind of service is provided, and to which consumers. During this registration process the users are often required to provide personal information, which they may not be comfortable to disclose (e.g., home address, phone number or credit card number).

This becomes a critical issue as users usually have no means of verifying whether the service provider can be trusted with the disclosure of such sensitive information. However, through semantic annotations, rule-oriented access control policies and automated trust negotiation, entities can incrementally build a trust relationship through which they will feel comfortable sharing selected private information.

Trust relationships may be established based on a set of digital credentials, which are disclosed between two entities to prove certain properties they own. Digital credentials attest a certain quality of the holder. They are issued and signed by a credential issuer and may bind the identity of the holder to a public key, attest a certain attribute of him or attest a relationship of the holder with regard to another entity. Credentials are integrity protected through the digital signature of the issuer. Credentials carry the public key of their holder what means that if an entity is able to sign a challenge with the private key associated to the public key present in the credential then that entity is the rightful holder of the credential (which may confer him a property). These credentials are the basis for a process in which trust relationships can be built from scratch, that is, between strangers following a process called *trust negotiation*.

During the trust negotiation process [22], trust is established gradually through disclosure of credentials and requests for credentials in an iterative and bilateral process. The requests for credentials are in fact authorization policies that have to be satisfied. A certain reached trust level through a negotiation between one client and a resource, may be regarded as an authorization decision whether or not to allow access to the resource. This approach distinguishes itself from the identity based access control in the following regards:

- Trust is established between previously unknown parties based on their properties, which are proved through credential disclosure.
- Entities (both service providers and consumers) can define policies in order to protect access to their resources (e.g., services and credentials).
- Authorization is no longer a one step decision, rather being established incrementally through a sequence of mutual requirements and disclosures of credentials.
- In the authorization process, more than two entities may be involved and their trusted credential issuer,

as requirements for a credential may determine a new negotiation with another entity, which at its turn may trigger another and so on.

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 where access is attempted, 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.

### 3.2. PEERTRUST Policy Language

Definite Horn clauses are the basis for logic programs [23], which have been used as the basis for the rule layer of the Semantic Web. The PEERTRUST language [24,25] for expressing access control policies is also based on definite Horn clauses, i.e., rules of the form

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

The semantics of the language is an extension of that of SD3 [26] to allow the set of all PeerTrust programs to be viewed as a single global logic program. We refer the interested reader to [26,27] for details on the semantics. In the remainder of this section, we concentrate on the syntactic features that are unique to the PEERTRUST language. We will consider only positive authorizations.

*References to Other Peers* The ability to reason about statements made by other parties is central to trust negotiation. For example, suppose that an online library requires a student id, issued by University of Hannover, before it gives a discount for buying a book. One can think of this as a case of the library *delegating evaluation* of the query “Is the requester a student?” to the University of Hanover (UniHann). Once UniHann receives the query, the manner in which UniHann handles it may depend on who asked the query. Thus UniHann needs a way to specify which party made each request that it receives. To express delegation of evaluation to another party, we extend each literal  $lit_i$  with an additional *Issuer* argument,

$$lit_i @ Issuer$$

where *Issuer* specifies the party who is responsible for evaluating  $lit_i$  or has the authority to evaluate  $lit_i$ . For example, the library’s policy for discounts might mention  $student(X) @ 'UniHann'$ . If that literal evaluates to true, this means that UniHann states that the requester is a student at the University of Hanover. Therefore, the previous example might be specified as

Library:  
 $applyDiscount(Book, X) \leftarrow$   
 $student(X) @ 'UniHann'$ .

where  $X$  represents the requester. For clarity, we prefix each rule by the party in whose knowledge base it is included.

The *Issuer* argument can be a nested term containing a sequence of issuers, which are evaluated starting at the outermost layer. For example, it is unlikely that UniHann will answer all the requests directly from the library so a more practical approach is to ask the requester to evaluate the query himself, i.e., to disclose his student id:

Library:  
 $applyDiscount(Book, X) \leftarrow$   
 $student(X) @ 'UniHann' @ X$ .

The library can refer to the party who asked a particular query by including a *Requester* argument in literals, so that we now have literals of the form

$$lit_i @ Issuer \$ Requester$$

Using the *Issuer* and *Requester* arguments, we can delegate evaluation of literals to other parties and also express interactions and the corresponding negotiation process between parties. Therefore, the final shape of the rule of the online library would look like

Library:  
 $applyDiscount(Book) \$ Req \leftarrow$   
 $student(Req) @ 'UniHann' @ Req$ .

*Local Rules and Signed Rules* Each party defines the set of access control policies that apply to its resources, in the form of a set of definite Horn clause rules that may refer to the RDF properties of those resources. These and any other rules that the party defines on its

own are its *local* rules. A party may also have copies of rules defined by other parties (credentials), 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 UniHann to prove that she is really a student:

```
Alice:
student(alice) @ 'UniHann'
  signedBy ['UniHann'].
```

In this example, the “signedBy” term indicates that the rule has UniHann’s digital signature on it. This is very important, as the library is not going to take Alice’s word that she is a student; she must present a statement signed by the university to convince the library. 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 party receives a signed rule from another party, the signature is verified before the rule is passed to the evaluation engine. Similarly, when one party sends a signed rule to another party, the actual signed rule must be sent, and not just the logic programmatic representation of the signed rule.

*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 [28], 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. In particular, (a subset of) the guards in a policy rule can be viewed as a query that describes the set of resources for which this policy is applicable. The query is expressed in terms of the RDF characteristics of the resources to which the policy applies. In this paper, access will be granted to a resource if a user can satisfy any one of the policy rules applicable to that resource. (This scheme can be extended to allow negative authorizations and conflict resolution.) For exam-

ple, if Alice will show her student credential only to members of the Better Business Bureau, she can express that policy as a guard that is inside her student credential but outside the scope of UniHann’s signature. This is expressed syntactically by adding these additional local guards between ← and “signedBy”.

```
Alice:
student(alice) @ 'UniHann' ←
  member(Requester) @ bbb @ Requester
  | signedBy ['UniHann'].
```

Both local and signed rules can include guards. PEERTRUST guards express the evaluation precedence of literals within a rule, and any rule whose guard literals evaluate to true can be used for continued evaluation. In contrast, guards in parallel logic programming systems usually introduce a deterministic choice of one single rule to evaluate next.

#### 4. Distributed Policy Based Negotiation for GRID Services Authorization

This section addresses some of the current Grid Security Infrastructure limitations by describing how PEERTRUST policies<sup>1</sup> can be integrated to accommodate a large, loosely coupled Grid environment. Semantic annotations may be used at each available service, specifying its access control requirements in a machine-understandable language and as a result, requiring only minimum human intervention for its access policy configuration and management. Self describing services in terms of their authorization policy constraints bring a major benefit to heterogeneous Grid environments. It allows automatic and autonomous negotiation for access granting based on both sides’ policies and dynamic fetching of credentials. Rule-based policy authorization schemes have been identified in [30] as a solution for the transparent resource sharing over the Grid, which we extend here with automatic credential fetching capabilities.

<sup>1</sup> Although we use the PEERTRUST language for policy specification in our examples and implementation, all the ideas presented in this paper are extensible to any other policy language with *delegation of authority* and *negotiation capabilities* [29].

In our initial example (see figure 2) we illustrated how different resources may be used in a Grid environment based on the Globus Toolkit 4.0. This scenario included only two domains: one administrated by the University “Politehnica” of Bucharest (UPB) and the other by the Navy Institute. All university resources had been setup in advance to easily interoperate and authorize university members. The collaboration with the Navy Institute had already been established, such that its scientific instruments could be used by the university members. The fact that resources are often expensive in terms of cost, maintenance and management normally results in more complex access policies as more parties become part of the collaboratory.

In addition, the identity based authorization policy used in the previous scenario does not scale well, and would soon put a burden on the policy administration. Furthermore, the job may use discovery and scheduling services to locate the best available resources, which would make it impossible to predict before hand what exact service instances would be used. This last example shows that one could benefit from jobs and resources that would both have the ability to independently gather credentials and acquire authorization.

Our solution proposes, on one hand that each grid service advertises its authorization requirements through access control policies and, on the other hand that each client specifies his credential disclosure policies and is able to query for resource’s access policies. When such policies are advertised, services and clients can negotiate, increasing incrementally their trust relationship based on satisfying the other party’s policies. The process of policy negotiation can be automated in a self describing environment, achieving a dynamic and versatile Grid environment where resources are securely shared.

We will change our initial scenario in order to accommodate the new capabilities highlighted above. Furthermore, we will assume that more administrative domains are involved and that each party is able to advertise its access control policies.

Alice is attending a Grid Conference organized by the Global Grid Forum (GGF) in another country. Her laptop has no Grid job submission capabilities but she would like to start the job for collecting the needed data so that her group can continue the experimental work. In addition, Alice can not log into the UPB Grid Portal because, for security reasons, it can only

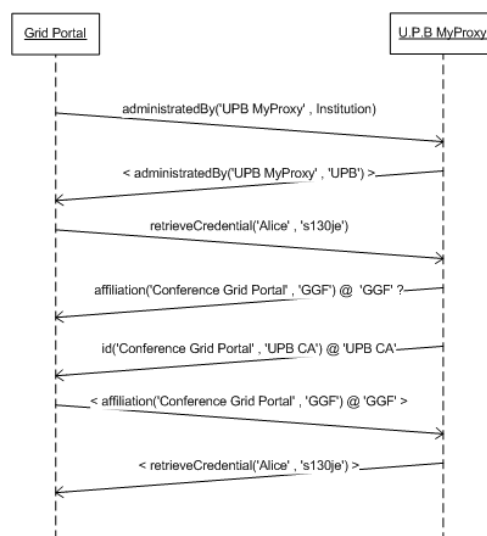


Fig. 3. Negotiation between the Grid Portal and UPB MyProxy

be reached from the internal university network. Fortunately, the conference provides its own policy enabled Grid Portal and Alice decides to try it out by requesting it to submit her job.

The policies regulating the access control at the Conference Grid Portal (CGP) specify that a requester must be registered at the conference and must provide the required authentication data before it allows a credential retrieval from a MyProxy server.

Conference Grid Portal (CGP):

```

allowRequest(MyProxy) $ Req ←
    validConfRegistration(Name, RegNumber, Req),
    authData(MyProxy, UsrName, Pwd) @ Req,
    permitted(MyProxy) |
    retrieveCredential(UsrName, Pwd) @ MyProxy.
validConfRegistration(Name, RegNumber, Req) ←
    registrationInfo(Name, RegNumber) @ Req |
    checkRegistration(Name, RegNumber).
permitted(MyProxy) ←
    administratedBy(MyProxy, Institution) @ MyProxy |
    registeredWithConference(Institution).
affiliation('Conference Grid Portal', 'GGF') @ 'GGF'
signedBy ['GGF'].
  
```

Alice sends a request to the portal to retrieve a credential from the UPB MyProxy Repository (that is the query `allowRequest('UPB MyProxy')`). The Grid Portal enforces its access control policies

and therefore verifies whether the user is registered at the conference by asking Alice to enter her name and conference registration number. Once Alice has provided this information, the portal checks locally if the registration data is correct<sup>2</sup>. In addition, the Grid Portal asks Alice for the username and password needed to access the MyProxy Repository and to retrieve a credential certifying that the job submission is made on Alice behalf.

Alice has all the information required and the conference portal validates its correctness. However, the grid portal requires the MyProxy repository to be administrated by an institution participating in the conference. Fortunately, UPB MyProxy is administrated by UPB which is registered with the conference as participant institution and does have a credential to prove this (and no policy protecting it). Therefore, the portal tries to retrieve the required credential sending the query `retrieveCredential('Alice','s130je')` to UPB MyProxy. However, UPB MyProxy does not allow just any entity to retrieve credentials and allows only interactions with a trusted entity:

```
UPB MyProxy:
retrieveCredential(UsrName,Password) $ Req ←
    valid(UsrName,Password),
    trusted(Req).
trusted(Req) ←
    affiliation(Req,'GGF') @ 'GGF' @ Req.
trusted(Req) ←
    id(Req,'UPB CA') @ 'UPB CA' @ Req.
administratedBy('UPB MyProxy','UPB') @ 'UPB'
signedBy ['UPB'].
```

As specified in the policy above, entities are trusted if they disclose either a certificate signed by the UPB Certification Authority attesting the identity of the requester or a proof of the requester being affiliated with Global Grid Forum (GGF). As shown in its policies, the conference portal owns a credential attesting its affiliation with GGF. In addition, this credential has no protecting policy and therefore is disclosed to UPB MyProxy upon its request. Finally, the trust negotiation process succeeds and the Conference Grid Portal

<sup>2</sup> For the sake of simplicity, we have omitted the facts of the policy with the data about e.g. registered participants

retrieves a delegated credential on Alice behalf. Figure 3 summarizes this negotiation.

Alice's research group needs the data as soon as possible so Alice instructs the Conference Grid Portal to query the Monitoring and Discovery Hierarchical Service (MDHS) in her home country to search the best Linux Cluster, in terms of memory size and number of available processors.

MDHS requires any requester to have a credential signed by one of the recognized state institutions. The Conference Grid Portal provides the credential delegated on Alice behalf by the UPB MyProxy Repository, and is authorized to query MDHS available information.

```
MDHS:
queryingAllowed() $ Req ←
    validCredential(Req).
validCredential(Req) ←
    id(Req,'UPB CA') @ 'UPB CA' @ Req.
validCredential(Req) ←
    id(Req,'Navy Ins. CA') @ 'Navy Ins. CA' @ Req.
...
```

The portal receives as a result of its query that the best available Linux Cluster belongs to the Research Center for Aeronautical Sciences (RCAS). Therefore, the portal initiates a new trust negotiation process for submitting Alice's job. In this negotiation, the RCAS Linux Cluster informs the Conference Grid Portal that only jobs acting on behalf of members of projects listed in the Ministry of Education (MinEdu) database are authorized for submission.

```
RCAS Cluster:
submit(Job) $ Req ←
    actingOnBehalfOf(User,Job),
    member(User,Project) @ 'MinEdu CAS' @ Req.
id('RCAS Cluster','MinEdu') @ 'MinEdu'.
signedBy ['MinEdu'].
```

By providing Alice delegated credential, the Grid Portal demonstrates that the job is acting on Alice behalf, but the portal does not yet have a credential proving that Alice is a member of a listed project. The Grid Portal tries to solve this situation and forwards a query to the Ministry of Education CAS server (MinEdu CAS), provides Alice's credential and retrieves an as-

sertion attesting that Alice participates in a project regarding signaling instrumentation. This credential is disclosed to the RCAS Linux Cluster and finally the Grid Portal is granted the rights for the job submission. Since the job needs to contact other resources, a proxy credential is also delegated by the Grid Portal, on behalf of Alice, to the RCAS Linux Cluster.

When Alice's job starts its execution, it has to retrieve the input data from the UPB Reliable File Transfer Service (UPB RFT). UPB RFT policies require the job to prove that it is acting on behalf of a UPB staff member.

UPB RFT Service:

```
retrieve(File) $ Req ←
  member(Req,'Staff') @ 'UPB CAS' @ Req |
  check(Rights,Req,File) @ 'UPB CAS'.
store(File) $ Req ←
  member(Req,'Staff') @ 'UPB CAS' @ Req.
```

Therefore, using its delegated credential, the job authenticates to the UPB CAS, demonstrates to be acting on Alice behalf, and retrieves a credential attesting that Alice is part of the university staff. Once this credential is released to the UPB RFT service, the service checks by itself, with the same UPB CAS if the required file can be accessed by Alice. UPB RFT acquires an assertion identifying the input file as belonging to Alice and in this way the file retrieval operation is allowed.

Now the job contacts the Navy Institute Wave Tank (with a query `access('Wave Tank')`) and discovers that it has to provide a certificate signed by either UPB or Navy Institute Certification Authority. By disclosing Alice delegated certificate, the job demonstrates to act on behalf of an entity certified by UPB CA. Further, the Wave Tank informs Alice's job that it should either provide a student credential or reveal Alice's roles at the university.

Wave Tank:

```
access(Resource) $ Req ←
  checkUser(Req,CA),
  checkLocalPolicies(Resource,CA).
checkUser(Req,'Navy Ins. CA') ←
  id(Req,'Navy Ins. CA') @ 'Navy Ins. CA' @ Req.
checkUser(Req,'UPB CA') ←
  id(Req,'UPB CA') @ 'UPB CA' @ Req, |
```

```
  validRole(Req,'UPB CAS').
validRole(Req,'UPB CAS') ←
  student(Req) @ 'UPB CAS' @ Req.
validRole(Req,'UPB CAS') ←
  role(Req,Role) @ 'UPB CAS' @ Req |
  Role = 'Researcher'.
checkLocalPolicies(Resource,CA) ←
  notInUse(Resource),
  usedResources(Hours,CA),
  timeLimit(Limit,CA),
  Hours < Limit.
bbbMember('Wave Tank','BBB') @ 'BBB'
  signedBy ['BBB'].
```

In this case, Alice had provided a policy to the job, stating that Alice roles are revealed only to entities that have proved their liability by disclosing a Better Business Bureau (BBB) membership credential.

Job:

```
role('Alice',Role) @ 'UPB CAS' $ Req ←
  member(Req,'BBB') @ 'BBB'.
```

The Wave Tank has no policy protecting its BBB credential so after its disclosure, the job contacts UPB CAS to retrieve Alice's roles. Since Alice is a researcher and assuming that the other local policies are fulfilled, the job is granted access to the Wave Tank.

The refined and corrected data generated by the Linux Cluster is saved using the UPB RFT Service, which allows file storage if the action is requested by a university staff member (see RFT policy above). The job has cached this credential (during a previous interaction with UPB RFT Service) and does not need to ask for it again from the UPB CAS. By revealing this proof of Alice university staff membership, the job is allowed to use the UPB RFT for storing its output files. As in our initial example, the credential delegated to the Research Center for Aeronautical Sciences is to expire so the RCAS Linux Cluster has to renew its credential by contacting UPB MyProxy. The policies enforced for renewing such a credential require a proof provided by the requester, that a previously credential was delegated and that it owns a credential signed by Ministry of Education.

UPB MyProxy:

```
renew(User,Credential) $ Req ←
```

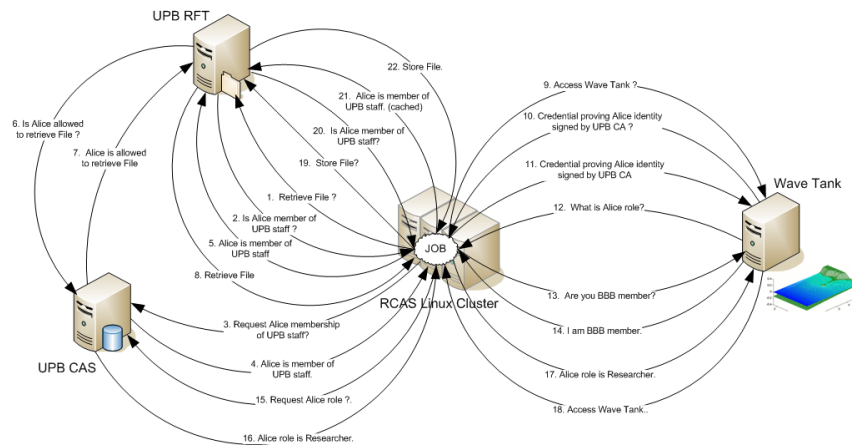


Fig. 4. Job Negotiations

previouslyDelegated(User,Credential) @ Req.  
id(Req,'MinEdu') @ 'MinEdu'.

Since RCAS Linux Cluster has the previously delegated credential and no policy regarding the disclosure of its Ministry of Education signed certificate, trust negotiation succeeds again and the job certificate is renewed.

This scenario shows that the job has been submitted with only one credential and dynamically negotiated authorization at each resource accessed. This negotiation involved learning from resources themselves which credentials are needed (specified in policies) and where to retrieve them from. This scenario is possible due to the grid services' ability to advertise policies, to indicate what credentials are needed and to indicate who the required issuer must be. Resources were shared with no implied previous interactions and with no further involvement of administrators to set policies for access control and credential disclosure.

## 5. Enhanced Authorization Characteristics

We have seen in the previous section how authorization decisions are made automatically based on each party's specified policies. The dynamic gathering of credentials is achieved because of the expressiveness of the policies that allows the resources to advertise both their requirements and how they can be satisfied. Our proposed authorization architecture comes with

a set of enhancements over the existing approaches, permitting autonomous entities to reach an agreement for authorization granting (if policies and credential disclosures permit it). We summarize all the characteristics of our authorization architecture and show why they are most suitable for a large scale, pervasive and heterogeneous distributed environment as the Grid.

*Distributed Authorization Mechanisms.* Resources define their own access control policies. As part of the authorization process, assertions regarding user's capabilities can be obtained through interrogation of other resources (e.g., repositories) based on delegation of authority. Users may retrieve assertions themselves from repositories indicated by the service being accessed, or they may require that the accessed service retrieves and proves at runtime certain properties. Moreover, local administrators can set their policies autonomously without needing to contact each other each time a new user needs to be authorized.

*Bilateral Policy Specification.* Both clients and service providers can specify policies to protect available credentials. Only if these policies are satisfied by the requesting party, access to the resources (e.g., credentials) is granted.

*Access is negotiated.* Users and services may have sensitive credentials that they do not want to disclose just to anybody. Access decisions are no longer a one step process as services and clients can now disclose

their credentials iteratively while increasing the level of trust until a final decision regarding authorization can be made. During the negotiation process several paths (different sets of disclosed credentials) can be followed based on client and service policies and strategies.

*Support for both credential push and pull model.* Clients can push to the server the set of negotiated credentials or indicate to the server where to retrieve client credentials. The service can also pull the client's capabilities from other services if those services are identified in access policies involving delegation of evaluation. For example the University Reliable File Transfer Service checks with the University CAS server if Alice is allowed to retrieve a certain file. Clients are themselves part of the pull model as service may direct them to repositories from where certain capabilities can be retrieved.

*Dynamic Credential Fetching.* Clients/Services no longer have to hardcode where to retrieve credentials from, or to expose unnecessary credentials as they can be requested and fetched at runtime based on their policies.

*Resource level policies.* The proposed architecture uses policies managed at the resource level, adhering to the Grid authorization model in which the resource provider has the ultimate control over the shared resources. Credentials might be gathered from different entities, pushed by the client or pulled by the service but the final authorization decision is made locally.

*Capability based authorization architecture.* Decisions regarding authorization are inferred based on user capabilities, relying on client properties and not on his identity. This gives the resource providers the extensibility and flexibility to enforce access granting policies at the resource level with minimal effort. In addition, we do not impose any restriction regarding the format of the credential as the policy language provides the abstraction of the actual mechanism. Therefore, attribute certificates, end entity certificates, signed assertions, etc. are all valid credentials.

*No previous trust relationships required.* Trusted authorities are not required from the start, as trust can

be built from scratch. It can, for instance, start with usage of self signed certificates just to establish a secure connection and then exchange through policies, information regarding who is trusted at each side and where to retrieve the required capabilities from. This feature could be most useful at the authentication level as different grid service may advertise who is considered trusted, and have clients automatically retrieve credentials from those trusted authorities.

*Explanation of the authorization decision.* The negotiation process can be traced at any moment to offer the client information about its status, where and why it has failed or how it evolved until access was granted. Using this trace the client can find out which credentials were required and could not be obtained.

## 6. Architecture and Implementation

In this section, we present an approach to integrate the extensions presented in the previous chapter within the recently released Globus Toolkit version 4.0 (GT 4.0). Our approach allows advanced access control mechanisms based on policies, dynamic negotiation for authorization and automatic fetching of credentials. One of our main design goals was direct integration with the grid services paradigms which resulted in an extension that is easily pluggable into any GT 4.0 compliant grid service or client. In the rest of this section, we present the technical details of our implementation (see also figure 5) as an extension to the current Grid Security Infrastructure [4]. Our implementation has been developed for the Globus Toolkit 4.0 Java Container, is entirely written in Java, and requires therefore Java based grid services and clients for a straightforward integration. A brief description of our work is also presented in [31].

The GT 4.0 Authorization Framework supports pluggable Policy Decision Points (PDP) [7]. A PDP is used as an authorization decision oracle that evaluates the client's request and returns an authorization decision. It uses information about the requester, the resource and the requested operation, and matches those with the available access policies. The returned decision can be either operation "granted" or "denied". Several PDPs can be specified allowing a chain

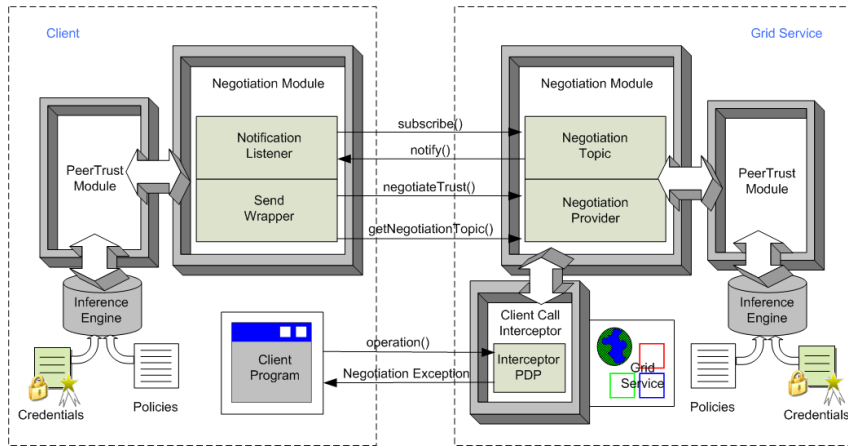


Fig. 5. High Level Architecture

of verifications to be performed. The final authorization decision is the conjunction of the results of all of them, that is, all of them must return operation “granted” in order to permit the client’s request. This setup introduces some limitations as it requires that the client satisfies all policies in the PDP-chain and does not allow simple disjunction. As we have seen in our previous discussions regarding the PEERTRUST capabilities, a service might also allow different possibilities for the access granting, where the client is able to choose which one to follow according to his own credential protection policies (e.g., choose the one where less sensitive information is revealed or those locally available).

Because of the GT4.0 limitation in the PDPs decision composition (no disjunction supported), we have developed a custom PDP, which is part of the *Client Call Interceptor* (check figure 5), and responsible only for protection against unauthorized calls. We have used the PEERTRUST language and implemented its logic outside this custom PDP to permit policy combinations both as conjunctions and disjunctions. Our custom PDP intercepts client operation invocations and allows or denies the calls according to the existence of a previous successfully completed negotiation. In addition, the PDP temporarily caches information about the authorized requesters and their access granted decisions, to permit successive client calls without the need to repeat the negotiation process. In summary, the Client Call Interceptor is responsible for authorizing the client invocations, for handing the au-

thorization process off to a negotiation module, and for checking for successfully completed negotiations.

A PDP is associated with a grid service through a security descriptor (a XML file pointed out in the service WSDL file). When an operation for that service is requested, the PDP is invoked automatically by the Globus Toolkit container with the operation name and the requester (client DN and certificates) as parameters. The PDP checks at the *Negotiation Module* whether a successful negotiation for the operation requested has been previously completed. If the answer is positive then the PDP grants the grid service operation invocation. If the answer is negative the PDP throws a negotiation exception to inform the client that a trust negotiation must take place which should be completed successfully before access is granted. In the latter case, the client Negotiation Module must start a negotiation with the homologous module on the grid service (with a query of the form `request('OperationName')`). This triggers the negotiation process, in which policies and credentials are exchanged (as described in section 3.1). The entire negotiation process is automated and requires no explicit user intervention.

The *Negotiation Module* is responsible for the negotiation management. It handles the communication with other parties using *Grid connectors* (implemented with the Globus Toolkit 4.0) and interfaces with the PEERTRUST *Module* (described later in this section). Whenever a negotiation succeeds the Negotiation Module caches the identity of the client and

the operation granted in order to avoid the same negotiation in (presumably) successive calls. The cached authorization decision should be invalidated once the shortest-lived credentials disclosed by the client have expired. Although we plan to support a wide variety of credentials, for the current implementation we only relied on the expiration time of the client proxy certificate (the one used by the client for authentication).

On the service side, the Grid Connectors are composed of two modules: the *Negotiation Provider* and the *Negotiation Topic*. These modules are plugged into the grid service, enhancing its functionality with policy based negotiation capabilities. The client can use the additional grid service's operations, implemented in the *Negotiation Provider Module*, for pushing policy requirements and credentials to the server side.

Negotiations are essentially asynchronous by nature, due to the fact that a request might, for instance, trigger a new policy negotiation with a third party and so on, which makes it difficult to predict when the request will finish its evaluation. Moreover, a client might want to continue its usual execution without having to wait for the answer from the server (e.g., in order to start in parallel several negotiations with different entities) and might just want to be informed asynchronously when each negotiation is finished<sup>3</sup>.

Asynchronous client notifications can be implemented through GT 4.0 support for WS-Base Notification [32] and WS-Topics [33], which standardizes asynchronous communications between consumers and providers. WS-Topics specifies how topics can be organized and categorized as items of interest for subscription. We associate each negotiation (triggered by the client request for a service operation) with a topic of interest (*Negotiation Topic*) through which messages can be delivered to the client side. The client must register with a certain topic of interest associated with a unique namespace (identifier) received through the invocation of the `getNegotiationTopic` function (exposed by the grid service through the *Negotiation Provider Module*). The registration is completed when the client subscribes to the topic associated with the returned namespace via a `subscribe` operation exposed by the notification producer (in our case the grid service). On the client side, the notification consumer

(implemented as a listening thread) exposes a notify function that the producer uses to send the notification each time the topic has changed. Using the notification mechanism the grid service can inform the client about its own requirements in terms of authorization.

The use of the GT 4.0 implementation of the notification paradigm requires the grid service to expose its state as a "Resource" in conformance with the Web Service Resource Framework [34]. "Resources" are used for storing a web service state from one invocation to another. Thus the only requirement for integrating our policy based architecture for grid service authorization is having the service implement such a "Resource". The addition of a resource to a grid service is straightforward, with minimal additions to its descriptor files and code (a simple implementation of an interface - Resource - with no methods). The "Resource" used by the service also has to implement the `TopicListAccessor` interface with only one operation `getTopicList`, which is used by the *Negotiation Provider* to store and remove negotiation topics.

A grid service describes its supported operations and their associated parameters through a so-called WSDL file. The `WSDLPreprocessor` namespace is a useful feature of the Globus Toolkit for writing WSDL code for a grid service. This namespace contains a tag "extends" which permits the inclusion of definitions in a grid service WSDL. Through this feature, one can easily include the description of the functionality of one service into another service. We have defined a special WSDL file that describes all the interfaces needed to perform our negotiations. Any grid service can use our negotiation capabilities by extending its WSDL description with our definitions. This allows a grid service to expose two additional operations: `getNegotiationTopic` and `negotiateTrust`, which are used to push messages to the grid service during the trust negotiation process. These two operations are implemented externally to the application-specific grid service code in our *Negotiation Provider Module*. This module is plugged into a grid service through the specification of a "provider" (`TrustNegotiationProvider`) in the grid service deployment descriptor (WSDD file). Providers are used to confer to grid services extra functionalities. GT4.0 support for notifications is also plugged into a grid service by means of two

<sup>3</sup> Note that a server might act as a client if it starts a new negotiation with a third party.

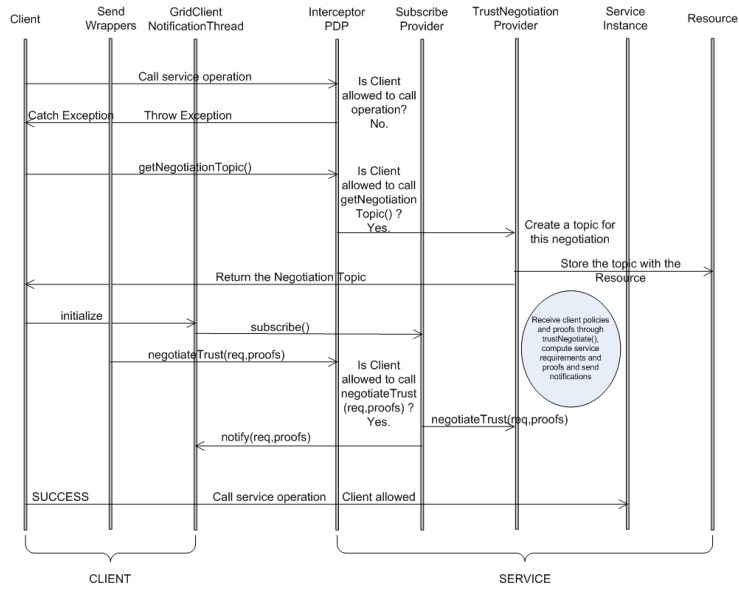


Fig. 6. Sequence Diagram

providers (delivered with GT4.0) configured in the same deployment descriptor (SubscribeProvider and GetCurrentMessageProvider). We emphasize that in order to confer the negotiation capabilities to any grid service we require small modification to its descriptors (configuration) files, but no modification to the application code itself.

For the client side, we have developed an API with a jar file to facilitate easy integration of trust negotiation capabilities with the client code<sup>4</sup>. Through the API, the client needs to set the grid service address and the namespace of the received negotiation topic, and must implement an interface (named SendWrapper) to communicate with the service with whom trust is negotiated. By implementing this interface, the client can also use an object/class for the automatic fetching of a SAML assertion from a CAS server. Our code includes the creation of a new thread (GridClientNotificationThread) which registers with the topic returned by the grid service and listens for notifications. When a notification is delivered, this thread processes it and passes it to the PEERTRUST Module. A sequence diagram which

shows the interactions between different elements on the client and server is depicted in figure 6.

So far we have introduced how the policy based negotiations can be integrated in the current GT4, but we have not described how the reasoning is performed. Queries for enforcement of policies are sent to the PEERTRUST Module, which controls the access to the Inference Engine. The Inference Engine is built on the Minerva Prolog engine<sup>5</sup> and reasons on policies and credential configured locally or retrieved during the negotiations undertaken. The Inference Engine answers to the PEERTRUST module queries indicating whether a request conforms to local policies (therefore allowing or disallowing the disclosure of credentials requested) or whether there are some extra conditions that must be first satisfied by the requester. The evaluation of a query is based on the (possibly) cooperative distributed computation of a proof tree. A proof tree represents the different paths on which the negotiation may evolve (see figure 7). This evaluation allows to include a proof in any answer returned to other parties. This proof contains the policies and credentials (possibly including third party credentials) required to

<sup>4</sup> This code can also be used by a service when it has to act as a client in order to perform negotiations with other services or to retrieve credentials.

<sup>5</sup> Minerva Prolog (<http://www.ifcomputer.com/MINERVA/>) provides a java-based prolog engine that allows for an easy integration.

attest the other party request (figure 7 shows an example of a proof).

Because our implementation uses an abstract representation of credentials and policies, it is able to use different kinds of formats (certificates, signed assertions, signed RDF statements, etc). Currently we support X.509 v3 Certificates which can carry holder attributes in the extensions. In addition, we have integrated the use of proxy certificates which carry SAML Assertions retrieved from a Community Authorization Service (CAS). SAML assertions can be wrapped in proxy certificates and express a Subject-Action-Object relationship which can easily be mapped to our policies as `Action(Subject) @ Object` (e.g., a CAS assertion containing “Alice-Member-EarthquakeProject” can be represented as

```
member('Alice')@'EarthquakeProject'  
signedBy['CAS'].)
```

## 7. Related Work

Our main contribution to the Grid authorization solution space is the presented trust negotiation process in which entities, distributed over the Grid, engage each other for access granting. During this process, entities express their access requirements and enumerate the required credentials with their associated third party issuers. As credentials are retrieved dynamically during the negotiations, the access policies fulfillment is established at runtime. Furthermore services and clients are true peers in the negotiation as both are able to protect their local resources (e.g. credentials and provided services) and to formulate requests for the required credentials from the other party.

One of the current trends in the Grid deployments is to move from identity-based to attribute-based authorization or role-based access control (RBAC). As mentioned before in this paper, the main reasons for this shift are the scalability issues associated with identity-based policies and the ease of administration through abstraction mechanisms as groups and roles. In this section we will give a brief overview of related approaches to Grid authorization and compare those with the one presented here. All the discussed

solutions identify the need for an authorization mechanism based on assertions like user capabilities, and not their identities. Through this mechanism, the policy for a larger number of entities can be managed more easily without the consistency issue of the mapping and remapping of identities to user accounts. However, despite the increased scalability, this approach still places the burden on the client side to retrieve and obtain the appropriate certificates for every service that is accessed. In addition, most authorization schemes are server policy centered, and no policy is enforced on the service itself by the client.

The *Virtual Organization Management Service* (VOMS) [17] supports attribute based access control for the Globus Toolkit Resource Allocation Manager (GRAM), which is responsible for job execution. Clients retrieve from the VOMS service an X.509 Attribute Certificates (AC) asserting the client’s groups and roles. This AC is subsequently embedded in a non-critical extension of a proxy certificate and communicated to the application service during the normal authentication process. After the authentication, the AC is extracted from the proxy certificate and validated, and the attributes are made available for the subsequent authorization decision evaluation.

The *PERMIS* [35] project’s main goal is the construction of an X.509 role based Privilege Management Infrastructure that accommodates diverse role oriented scenarios. PERMIS consists of two subsystems: the privilege allocation subsystem which issues X.509 Attribute Certificates (ACs) and stores them in LDAP directories for later retrieval, and the privilege verification subsystem which pulls the user certificates and the policies regarding user roles from a pre-configured list of LDAP directories. Clients can also obtain their AC and push those certificates with the request to the privilege verification subsystem.

The system for *Privilege Management and Authorization* (PRIMA) [16] achieves authorization in a Grid environment through the management and enforcement of three features: a set of XACML encoded *privileges* contained in X.509 Attribute Certificates describing the user fine-grained access rights, a *dynamic policy* computed on the user specific request and the resource configured access policy, and a native *dynamic execution environment* restricting user rights to those of the acknowledged privileges. In PRIMA, the administrators and stakeholders are responsible for the

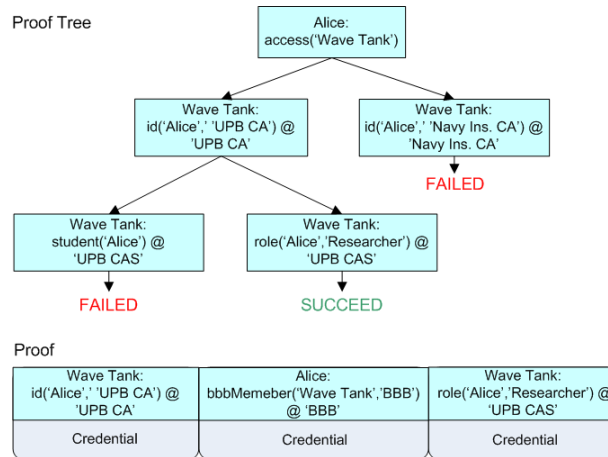


Fig. 7. Proof and Proof Tree computed on the Wave Tank side - see the negotiation for job access to the Wave Tank (section 4)

configuration of the resource access policies and the externalization of the access rights as privileges to the users. Before accessing a resource, the user selects the privileges to be disclosed and presents them to the resource's Policy Enforcement Point (PEP). From the user specific request and his validated privileges, the PEP constructs a dynamic policy and forwards it together with an XACML authorization request to the resource's Policy Decision Point (PDP) for an authorization decision evaluation. The PDP responds to the PEP with an XACML authorization decision (permit or deny) rendered from the dynamic policy and the resource's own access policy, plus a set of obligations that are used to set up the execution environment (file access permissions, user quotas and network access [36]). Access is granted only if the PDP XACML authorization decision was a permit and the PEP is able to meet the PDP obligations.

The *Community Authorization Service* (CAS) is an authorization services where the client can ask for a token that would include the access rights needed to invoke a request at a remote service. The CAS maintains a database of policy rules for the VO and returns a signed SAML authorization decision assertion that reflects the decision result of the evaluated client request. The assertion contains the identity of the user and the actions allowed on the resource. The assertion is embedded by the client into a user proxy certificate and pushed to the service with the request. Before access is granted, the service will extract and validate the assertion, verify whether it applies to the client's

request, and verify that the assertion was signed by a CAS identity that is allowed to issue such a authorization statement.

*GridShib* [37] is a recently started project with the goal to integrate Shibboleth [38] and the Globus Toolkit. Shibboleth is a privacy-preserving attribute authority service developed by the Internet2 community for cross-organizational identity federation. The integration project yields a Globus Toolkit runtime module that will transparently retrieve attribute assertions from the Shibboleth service for the requester, which are used for the subsequent authorization decision evaluation. The protocol used is the SAML attribute query protocol, and the attributes are communicated in SAML attribute assertions.

*XPOLA* [39] provides a capability-based authorization infrastructure, enabling a user-level, peer-to-peer collaborative Grid environment. In this model, the service provider creates a set of capability tokens for all the possible clients that are allowed to access its resources and makes those available to the clients for retrieval. Before the invocation of a service request, the client will retrieve the applicable capability, and will push that together with the request. The service provider will validate the capability and ensure that it applies to the client's requested operation, after which access is granted. The capabilities are based on the SAML authorization decision assertions.

Note that all the mentioned authorization solutions in this section rely on fixed configurations of trust roots, on expected knowledge of required credentials

and on expected knowledge where such credentials can be retrieved. Their deployment is therefore inflexible and policy changes are expensive. Note also that none of the described solutions deal with the policy enforcement on the client side.

In contrast, our architecture deploys user attributes for access granting, but in addition also permits the specification of complex relations between user attributes in the access policy rules. The peers are able to discover each other's policies incrementally through negotiation, and automatically fetch the credentials required to fulfill the policy constraints. Furthermore, client and server are considered peers in the trust negotiation as both have access policies that are enforced. In our design there are no static or pre-configured lists of user rights and attribute authorities, and no assumptions are made about the client's awareness of the access requirements prior to the service invocation. Lastly, the authorization process is fully automated and agreed upon through negotiations undertaken at runtime.

## 8. Standardization

There are a number of available and emerging standards that we have based our current implementation on and that we hope to further leverage in our future efforts.

Our current implementation uses :

- the standard webservice SOAP [40] over HTTP protocols as integrated in the GT4, and as standardized by W3C and OASIS.
- the webservice security standards that provide for message integrity, privacy and authentication, which are also integrated in the GT4, standardized by the Webservices Security TC in OASIS [41], and profiled by WS-I [42].
- SAML 1.1 [14] compliant assertions for the attribute credentials from CAS. The SAML message formats and protocols are standardized in the Security Services TC at OASIS.
- X.509 End Entity and Proxy Certificates for identity and authorization assertions as standardized by the PKIX working group at the IETF

Furthermore, the PeerTrust language is used to express the policy constraints, and so far no standard-

ized policy language seems available that includes all the required features to support our policy negotiation requirements.

There are a number of emerging specifications and standards that we are following closely to see if we can leverage their specific strength and adoption for different components of our framework:

- XACML [9]: The eXtensible Access Control Markup Language (XACML) TC at OASIS has developed the specification for a feature rich authorization policy language. Unfortunately, the language does not include facilities for delegation of rights, policy advertisement and capabilities matching. Ideas to modify and extend the language to accommodate these feature are on the roadmap but still in an early stage.
- WS-Agreement [43]: The Grid Resource Allocation Agreement Protocol (GRAAP) working group at the Global Grid Forum (GGF) is developing the Web Services Agreement Specification (WS-Agreement), a Web Services protocol for establishing agreement between two parties, such as between a service provider and consumer, using an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties. We are investigating whether we could benefit from adopting their proposed negotiation template.
- WS-Policy [44]: A number of vendors have published the ws-policy-\* set of draft specifications. The Web Services Policy Framework (WS-Policy) provides a general purpose model and a corresponding syntax to describe and communicate the policies of a Web service. WS-Policy defines a base set of constructs that can be used and extended by other Web services specifications to describe a broad range of service requirements, preferences, and capabilities. We are investigate how our negotiation protocol could benefit from the ws-policy components, which could help with interoperability once those specs have been standardized and widely adopted.

It is encouraging to see that some of the requirements for policy advertisement and capability matching that we have identified and addressed in our negotiation framework, are recognized. As stated we will follow the developments in the XACML, SAML and WS-Policy efforts closely, and hope to leverage on

some of the policy primitives that may result from those effort in the negotiation protocols that we have developed.

## 9. Conclusions and Further Work

This paper presents an architecture for a sophisticated authorization framework based on Semantic Web technologies. The resulting framework has features that address scalability and that simplify the policy management of complex deployments spanning multiple administrative domain - features that make our framework a compelling solution for specifically Grid environments.

The main framework features are:

- Policy-based authorization relying on parties' properties rather than their identities.
- Self-describing resources for access requirements.
- Dynamic negotiation for service authorization.
- Automatic credential fetching from credential repositories.
- Both server and client authorization policies.

The implementation of the framework is based on and integrated with the recently released Globus Toolkit 4.0 as an extension of the Grid Security Infrastructure (GSI). In our implementation we use the PEERTRUST language [24] and its Prolog-based reasoning engine, which provides support for the verification of X.509 v3 Certificates. Our implementation has been tested successfully through prototype applications that were able to negotiate the access to different Grid services and were able to automatically fetch required credentials from a CAS server. In addition, our implementation is distributed as open-source such that anyone can generate the jar files and use our APIs, which allows for an easy integration of our negotiation-based authorization framework with newly developed GT 4.0 grid services and client.

In the near future, we plan more experiments to study the performance cost induced by our policy negotiation approach. Although we know in advance that our negotiation requires more round-trips and is therefore more expensive than conventional approaches, it will be a trade-off with the time and efforts needed for hard-coded configurations and for out-of-band policy information exchange without negotiation capabil-

ities. We will also investigate the most simple scenario where only a two step negotiation would be required to query resource access policies and their subsequent fulfillment, which would give us a lower-limit performance mark.

Our future research will also focus on online credential repositories to improve the automatic fetching of credentials and user capabilities. We have already integrated with the CAS service for credential retrieval, and we plan to do the same with MyProxy repositories. Unfortunately the current MyProxy implementation does not support a webservices interface, nor is there a standardized interface defined yet for credential repositories and federation services. We plan to develop some wrappers that will give us the required capabilities through webservices protocols. Currently, the credentials express only user attributes (e.g. group or role) or a simple user-resource-action relation, and not more complex relations (e.g. signed rules), which would be desirable for the more complex scenarios we described. We plan to investigate whether more expressive languages, like XACML, could be used to define the more complex relations as signed policy statements.

As portals are popular interfaces to the Grid resources, we intend to develop a trust negotiation enabled Grid Portal to facilitate job submission and initial credential fetching through a web-like interface.

Another area of research is the possible utilization of our traceable negotiation process for accounting purposes, like billing and audit. Our infrastructure already supports the recording of the negotiation steps at each resource involved. The iterative process of requests for and disclosures of credentials may be extended to include the negotiation for pricing and exchange of payment until an agreement is reached.

Lastly, we plan the integration of a more expressive language than PEERTRUST into our framework to allow, for example, the execution of arbitrary operations when certain policies are satisfied (e.g., logging information). We plan to do so in the context of the European Union NoE REVERSE [45] project, in which the PROTUNE language [46] is identified as the future European Semantic Web policy language.

## References

- [1] Globus Toolkit, <http://www.globus.org>.
- [2] The Open Grid Services Architecture OGSA, version 1.0.
- [3] Globus Toolkit 4.0, <http://www.globus.org/toolkit/docs/4.0/>.
- [4] Grid Security Infrastructure, <http://www.globus.org/security/overview.html>.
- [5] R. Housley, T. Polk, W. Ford, D. Solo, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 3280 (2002).
- [6] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist, X.509 Proxy Certificates for Dynamic Delegation, in: 3rd Annual PKI R&D Workshop, 2004.
- [7] GT 4.0 Authorization Framework, <http://www.globus.org/toolkit/docs/4.0/security/authzframe/>.
- [8] OGSA Authorization Working Group, <https://forge.gridforum.org/projects/ogsa-authz>.
- [9] OASIS eXtensible Access Control Markup Language (XACML) TC, [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=xacml](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).
- [10] J. Novotny, S. Tuecke, V. Welch, An Online Credential Repository for the Grid: MyProxy, in: Symposium on High Performance Distributed Computing, San Francisco, 2001.
- [11] Condor-G, <http://www.cs.wisc.edu/condor/condorg/>.
- [12] L. Pearlman, C. Kesselman, V. Welch, I. Foster, S. Tuecke, The Community Authorization Service: Status and Future, in: Proceedings of the Conference for Computing in High Energy and Nuclear Physics, La Jolla, California, USA, 2003.
- [13] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, in: International J. Supercomputer Applications, 2001.
- [14] OASIS Security Services (SAML) TC, [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=security](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security).
- [15] W. Allcock, I. Foster, R. Madduri, Reliable Data Transport: A Critical Service for the Grid, in: Building Service Based Grids Workshop, Global Grid Forum 11, 2004.
- [16] M. Lorch, D. Adams, D. Kafura, M. Koneni, A. Rathi, S. Shah, The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments, in: Proceedings of the 4th Int. Workshop on Grid Computing - Grid 2003, Phoenix, AZ, USA, 2003.
- [17] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lőrentey, F. Spataro, VOMS: An Authorization System for Virtual Organizations, in: Proceedings of the 1st European Across Grids Conference, Santiago de Compostela, 2003.
- [18] S. Farrel, R. Housley, An Internet Attribute Certificate Profile for Authorization, RFC3281.
- [19] P. A. Bonatti, N. Shahmehri, C. Duma, D. Olmedilla, W. Nejdl, M. Baldoni, C. Baroglio, A. Martelli, V. Patti, P. Coraggio, G. Antoniou, J. Peer, N. E. Fuchs, Rule-based Policy Specification: State of the Art and Future Work, Project Deliverable D1, Working Group I2, EU NoE REVERSE (sep 2004).
- [20] H. Boley, B. Grosz, M. Sintek, S. Taet, G. Wagner, RuleML Design, <http://www.ruleml.org/> (2002).
- [21] D. Olmedilla, R. Lara, A. Polleres, H. Lausen, Trust Negotiation for Semantic Web Services, in: 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), Vol. 3387 of Lecture Notes in Computer Science, Springer, San Diego, CA, USA, 2004, pp. 81–95.
- [22] W. H. Winsborough, K. E. Seamons, V. E. Jones, Automated trust negotiation, DARPA Information Survivability Conference and Exposition, IEEE Press, 2000.
- [23] J. W. Lloyd, Foundations of Logic Programming, 2nd Edition, Springer, 1987.
- [24] R. Gavriloiu, W. Nejdl, D. Olmedilla, K. E. Seamons, M. Winslett, No Registration Needed: How to Use Declarative Policies and Negotiation to Access Sensitive Resources on the Semantic Web, in: 1st European Semantic Web Symposium (ESWS 2004), Vol. 3053 of Lecture Notes in Computer Science, Springer, Heraklion, Crete, Greece, 2004, pp. 342–356.
- [25] PeerTrust Project, <http://sourceforge.net/projects/peertrust/>.
- [26] J. Trevor, D. Suci, Dynamically Distributed Query Evaluation, in: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Santa Barbara, CA, USA, 2001.
- [27] W. Nejdl, D. Olmedilla, M. Winslett, PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web, technical Report (Oct. 2003).
- [28] K. Ueda, Guarded horn clauses, in: Logic Programming '85, Proceedings of the 4th Conference, LNCS 221, 1986, pp. 168–179.
- [29] K. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, L. Yu, Requirements for policy languages for trust negotiation, in: POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02), IEEE Computer Society, 2002, p. 68.
- [30] J. Basney, W. Nejdl, D. Olmedilla, V. Welch, M. Winslett, Negotiating Trust on the Grid, in: 2nd WWW Workshop on Semantics in P2P and Grid Computing, New York, USA, 2004.
- [31] I. Constandache, W. Nejdl, D. Olmedilla, Policy Based Dynamic Negotiation for Grid Services Authorization, in: Proceedings of the Semantic Web and Policy Workshop, 2005, pp. 55–67.

- [32] Web Services Base Notification, <http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf>.
- [33] Web Services Topics, <http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.pdf>.
- [34] Web Services Resource Framework, [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wsrf](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf).
- [35] D. Chadwick, O.Otenko, The PERMIS X.509 Role Based Privilege Management Infrastructure, in: 7th ACM Symposium on Access Control Models and Technologies, 2002.
- [36] M. Lorch, D. G. Kafura, The PRIMA Grid Authorization System., *J. Grid Comput.* 2 (3) (2004) 279–298.
- [37] V. Welch, T. Barton, K. Keahey, F. Siebenlist, Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration, in: 4th Annual PKI R&D Workshop, 2005.
- [38] Shibboleth Project, Internet2, <http://shibboleth.internet2.edu>.
- [39] L. Fang, D. Gannon, F. Siebenlist, XPOLA - An Extensible Capability-based Authorization Infrastructure for Grids, in: 4th Annual PKI R&D Workshop, 2005.
- [40] Simple Object Access Protocol (SOAP), <http://www.w3.org/TR/soap/>.
- [41] OASIS Web Services Security (WSS) TC, [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wss](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss).
- [42] Web Services Interoperability Organization, <http://www.wsi.org/>.
- [43] Grid Resource Allocation Agreement Protocol WG, <https://forge.gridforum.org/projects/graap-wg>.
- [44] Web Services Policy Framework, <http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polfram/>.
- [45] REVERSE Project. I2 group on policy language, enforcement, composition, <http://reverse.net/I2/>.
- [46] P. A. Bonatti, D. Olmedilla, Driving and Monitoring Provisional Trust Negotiation with Metapolicies, in: 6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005), IEEE Computer Society, Stockholm, Sweden, 2005, pp. 14–23.