A Multipolicy Authorization Framework for Grid Security

Bo Lang,[1,2] Ian Foster,[1,3] Frank Siebenlist,[1,3] Rachana Ananthakrishnan,[1]Tim Freeman[1,3]
[1] Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL
[2] Beihang University, Beijing, China
[3] University of Chicago, Chicago, IL

Abstract

A Grid system is a Virtual Organization that is composed of several autonomous domains. Authorization in such a system needs to be flexible and scalable to support multiple security policies. Basing on the Web Services security specifications such as XACML, SAML, and the special security needs of the Grid computing, we have constructed an authorization framework in the Globus Toolkit 4 that can support multiple policies. This paper describes the concepts of our design and introduces the structure and the components of the authorization framework. To show the flexibility and scalability of the framework, we introduce a new blacklist/whitelist-based authorization mechanism that can be seamlessly integrated into the framework.

Keywords Authorization Framework, Grid Computing, Globus Toolkit, Blacklist/Whitelist-based Authorization

1.Introduction

Grid is a new kind of distributed computing technology. A Grid system is a virtual organization comprising several independent autonomous domains.[1] The security of the Grid system should provide the same protection that conventional

systems provide, including establishing the identity of users or services (authentication), protecting communications (encryption/decryption), determining who is allowed to perform what actions (authorization), and recording the important operations processed by the systems(auditing). At the same time, it needs to meet the special security requirements of Grid systems.[2] Authorization is an important part of Grid systems, in which every domain may have its own policy and may change its policy dynamically. Hence, the authorization mechanism of Grid computing platform needs to support multiple security policies and needs to have the flexibility to support dynamic changes in security policies.

The Globus Toolkit is a fundamental enabling technology of the Grid. The security functionality of Globus is called the Grid Security Infrastructure (GSI).[2,3] From version 1 in 1998 to the 2 release in 2002 and now the 4 release based on new open-standard Grid services, GSI has been developing rapidly. In GT1, GSI mainly provided message protection and authentication; it used public key cryptography and was based on the authentication protocol defined by the Secure Socket Layer (SSL). [4] In GT2, GSI introduced X.509 proxy certificates to support dynamic creation of computing entities and provided Community Authorization Service (CAS) to implement access control in dynamic created overlaid trust domains.[3,5] In GT3, the Grid technology worked with the emerging Web services technology; the result was the OGSA (Open Grid Services Architecture).[6] In GT3, GSI3 uses the powerful features of OGSA and Web Services security. Security functionalities are defined as OGSA services to allow them to be located and used as needed by applications. Some

of the Web Services security specifications are used to allow security messages and secured messages to be transported, understood, and manipulated by standard Web Services tools and software. In GT4, additional Web Services security specifications are implemented, and distinct WS and pre-WS authentication and authorization capabilities are provided.[7] GT4 implements the WS-Security standard and the WS-SecureConversation specification to provide protection for SOAP messages; it constructs an authorization framework that allows for a variety of authorization policies, and supports a callout to an external authorization service via the SAML protocol. The authorization mechanism of GT4 is constructed based on the security requirements of Grid systems and the Web Services security standards—the OASIS XACML (eXtensible Access Control Markup Language) and SAML (Security Assertion Markup Language). The framework is flexible, enabling dynamic supporting of multiple security policies and thus satisfying Grid computing authorization requirements.

Section 2 of this paper introduces the XACML specification, which is the basis of our authorization framework. Section 3 describes the design concepts, the structure, and the components of the authorization framework. Section 4 discusses the design and implementation of the blacklist/whitelist-based authorization mechanism that can be integrated into the framework. Section 5 summarizes our work and the advantages of the new framework.

2. Theoretical Basis of the GT4 Authorization Framework

GT4 implements the WSRF specification, a convergence of Grid and Web

Services technology. The Web services security specifications include WS-Policy and XACML, which express the Web services security policy; WS-Security and SAML, which define standard formats for security token exchange; and WS-SecureConversation and WS-Trust, which define standard methods for authentication and establishment of security contexts and trust relationships. GT4 uses the XACML authorization model, an important Web services access control standard.[8]

XACML defines an access control framework and the data flow model of the framework components, as shown in Fig. 1.

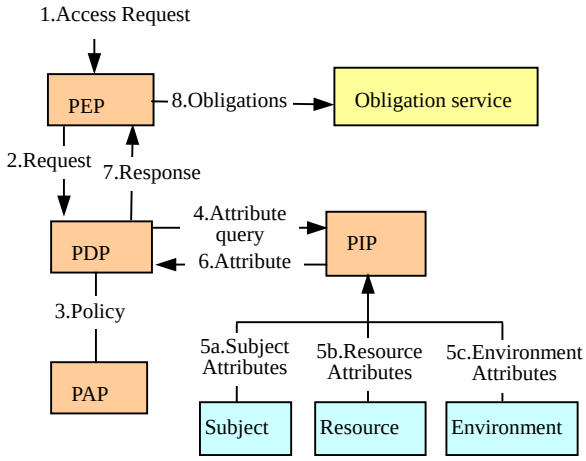

Fig. 1 XACML Data-Flow Diagram

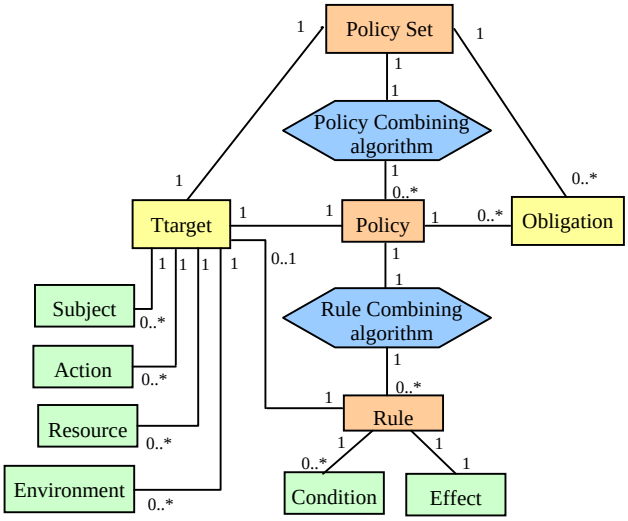

Fig. 2 XACML Policy Language Model

The access control framework mainly contains PEP (Policy Enforcement Point), PDP (Policy Decision Point), PIP (Policy Information Point), and PAP (Policy Administration Point). The PEP intercepts the access requests from users and sends the requests to the PDP. The PDP makes access decisions according to the security

policy or policy set written by PAP and, using attributes of the subjects, the resource, and the environment obtained by querying the PIP. The access decision given by the PDP is sent to the PEP. The PEP fulfills the obligations and either permits or denies the access request according to the decision of PDP.

XACML also defines a policy language. The policy model is shown in Fig. 2. The main components of the model are the rule, the policy, and the policy set. A rule is the most elementary unit of the policy and can be evaluated on the basis of its contents. The main components of a rule are as follows:

- A target that defines the set of resources, subjects, actions and environment
- An effect that indicates the consequence of the satisfied rule
- A condition that further refines the applicability of the rule

Rules are combined into a policy, which comprises four main components: a target, a rule-combining algorithm, a set of rules, and obligations. A policy set comprises four main components: a target, a policy-combining algorithm, a set of policies, and obligations. The rule-combining algorithm specifies the procedure by which the results of evaluating the component rules are combined when evaluating the policy. The policy-combining algorithm has a function similar to that of the rule-combining algorithm. Obligations are the actions that must be performed by the PEP in conjunction with the enforcement of an authorization decision; obligations are the mechanism for achieving finer-level access control.

3. The GT4 Authorization Framework

The convergence of Grid and Web services introduces both new opportunities and

new challenges for Grid security. On the one hand, these specifications have provided standard and interoperable methods for Grid security. On the other hand, in order to establish an authorization mechanism suitable for Grid computing, these specifications may also need to be extended or changed to some extent, since Grid has its own special application requirements.

In a Grid system, each domain has its own security policy, such as the grid-mapfile, ACL (Access Control List), CAS, SAML authorization decision assertions, and XACML policy statements. Hence, the GT4 authorization framework needs to support multiple security policies and also needs to be flexible, so that it can be changed easily for different application environments. These special authorization requirements are not addressed in the XACML specification.

Based on the XACML specification and the Grid access control requirements, we designed and implemented the GT4 authorization framework.

3.1 GT4 Authorization Framework Architecture

The GT4 authorization framework implements SAML and uses the XACML model, as shown in Fig. 3. It is composed of a PEP, PDPs, and PIPs:

- For each existing authorization policy, the framework constructs a PDP for evaluating that kind of policy. Four types of decision may be returned by each PDP: permit, deny, not applicable, and indeterminate. The Master PDP is responsible for coordinating the PDPs to render a final decision. It combines the decisions returned by each PDP through a combining algorithm.

- The PEP intercepts the user's request and executes the authorization decision received from the master PDP. The Master PDP and the PEP are collectively called the authorization engine of the framework.
- The PIPs are information collection points that collect attributes about various entities related to the authorization evaluation. A subset of PIP, referred to as Bootstrap PIPs, collect information only about the request, such as the peer subject, the requested action, and the resource. An example of one such PIP, is the X509BootstrapPIP, which extracts the subject DN of the peer from the X509 certificate.

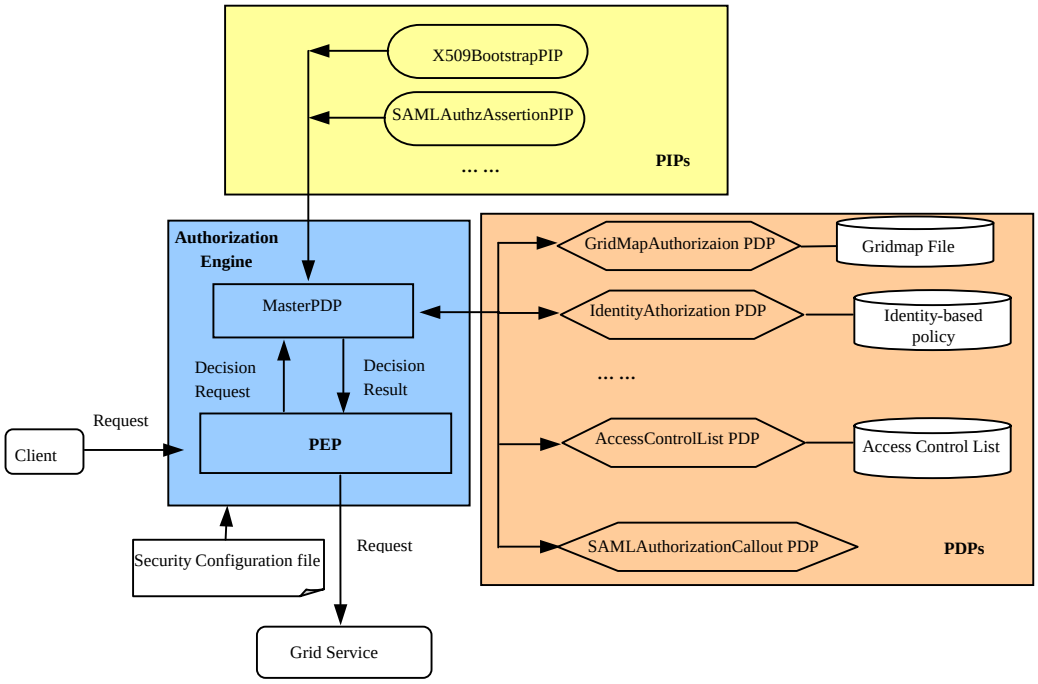

Fig. 3 GT4 Authorization Framework

In the authorization framework, the collection of Bootstrap PIPs, PIPs, and PDPs is referred to as an authorization chain. An authorization chain can be configured through the security configuration file or programmatically at the resource, service,

and container level. The level of authorization chain to be used is determined in the following order: the resource, the service, and the container: if the resource level is not configured, the framework looks for a service level configuration and if that is not configured it looks for a container level configuration.

When a request of the Grid resource comes, the PEP in the authorization engine intercepts it and sends a decision request to the master PDP. The master PDP collects information needed by calling the Bootstrap PIPs and other PIPs and then invokes the corresponding PDPs with the request and the information collected. The PIPs and the PDPs used are all specified in the security configuration file. When the master PDP receives the decisions returned by each PDP, it combines the decisions, using a policy combination algorithm to render a final decision, and returns the decision to the PEP. The PEP then executes the decision, either denying or permitting the request.

3.2 The PDP of the Authorization Framework

The PDP is the core of the authorization framework. For GT4 we designed a multipolicy framework to dynamically support multiple security policies and provide several PDPs that implement specific policies.

3.2.1 The Multipolicy Framework

One main challenge of the GT4 authorization framework is to support multiple policies. To achieve this goal, we built a multipolicy framework based on object-oriented technology. The framework is shown in Fig. 4. Because every policy essentially needs its own custom decision evaluator that understands the intrinsic semantics of the policy expressions, it is necessary to encapsulate the policy into an

independent PDP. At the same time, we abstract the common characteristic of the policies and define an abstract PDP. The PDP abstraction (the PDP class in Fig. 4) defines a common interface that can be used to interact with the PEP or with other PDPs. This common interface uses the XACML request context interface, which essentially presents the decision request as a collection of attribute values for the subject, resource, and action. Each specific policy is a subclass of the PDP abstraction, which implements the common interface inherited from PDP with its own policy and evaluation mechanism.

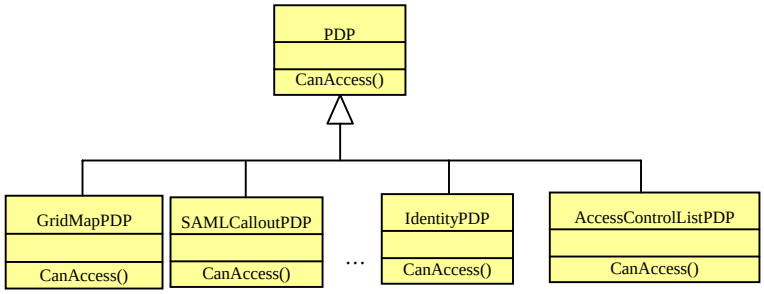

Fig. 4 GT4 Authorization Policy Framework

At run time, a separate Master PDP is used to create instances of the mechanism-specific PDPs specified in the security configuration file. The PDP instances construct a PDP chain. The Master PDP collects information about the request and calls the PDPs in the chain, combines the decisions from all the different PDP instances, and then renders a single decision reflecting the overall evaluated policies.

Since the policy framework is object oriented, it is scalable and flexible, which means that new policies can be added to the framework just by inheriting the PDP

class and that the existing policies can be removed and modified at any time. Also, since PDP instances are queried through the same interface and since the mechanism-specific details of the PDPs are all hidden behind the public interface, a change to the policy framework has no effect on the Master PDP: it can cooperate with any specific PDPs designated by the security configuration files. This multipolicy framework thus provides users of GT4 with an authorization mechanism that is flexible and scalable and can support multiple different policies.

3.2.2 The PDPs in GT4

In Grid systems, there are several frequently used simple authorization policies or mechanisms, we provided PDPs that implement these existing policies. There are also some authorization systems developed by others that can be used in a Grid system, such as Shibboleth,[9] Virtual Organization Management Service (VOMS),[10] and X.509 Role Based Privilege Management Infrastructure (PERMIS).[11] Therefore, we established a PDP for integrating other authorization systems through the SAML assertions. Some of the PDPs are as follows:

- AccessControlList PDP: implementing the access control list mechanism, which renders its decision by consulting a local user privilege file.
- GridMapAuthorizaion PDP: Looking at a Gridmap file to determine whether a requestor can access the service.
- HostAuthorization PDP: Checking whether the requestor has the specific host identity configured in the PDP.
- IdentityAuthorization PDP: Checking whether the requestor has the specific

identity configured in the PDP.

- SAMLAuthorizationCallout PDP: Integrating third-party authorization systems. It contacts an authorization service using the SAML authorization assertions to express the requests and the responses. The request can be permitted only when the authorization service returns a positive decision.

The Master PDP uses a combining algorithm to combine the decisions returned by each PDP. The algorithm can be configured in various ways. The following are examples of the algorithm:

- Deny override

  If a deny is returned by any PDP in the chain, the final decision will be deny. If no PDP in chain returns deny, the decision will be permit.

- Permit override

  If a permit is returned by any PDP in the chain, the final decision will be permit. If no permit decision chain is found, the decision will be deny.

- First applicable

  If a permit or deny is returned by any PDP, the decision is returned, and the rest of the chain will not be evaluated.

3.3 Attributes Collection

The authorization framework of GT4 is a kind of attributes-based access control.[12] Many policies use the attributes of the requestor, the service, the resource, the action, and the environment. Hence, it is also important that the framework build an effective mechanism to collect the attributes when making authorization decisions.

The attributes-collecting process is shown in Fig. 5.

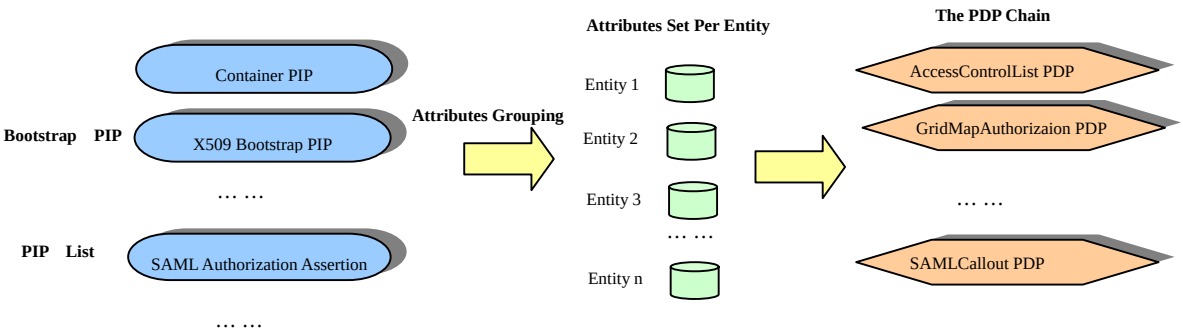

Fig. 5 Attributes Collection in GT4

When collecting the authorization information, the Container PIP is first invoked to collect attributes inherent to the framework, such as the service name and the operation name. Next, the Bootstrap PIPs are invoked to collect information about the request; usually the X509 Bootstrap PIP is invoked before any other Bootstrap PIP configured. Then, other PIPs are invoked in the configured order.

Each PIP might return a normalized representation of the collected attributes. The attributes then are grouped as a single set of attributes per entity and are stored, so that the PDPs in the PDP chain can access them when evaluating their policies.

4. Blacklist/Whitelist and Its Integration with GT4 Authorization

Blacklist and whitelist mechanisms are simple and well known in the security area. A blacklist is a list of particular entities identified by domain names, email addresses, or other attributes of the entity. The entities listed in a blacklist are considered dangerous or damage causing and are denied entry to the infrastructure they are trying to penetrate. Common examples of traditional blacklist solutions are anti-virus, anti-spyware software and email spam-preventing modules.[13] Whitelists

are the opposite, lists of entities that are allowed to use a system or service. The most common examples of whitelist solutions are email systems in which users create a list of authorized addresses from which they can receive mail.

The most obvious advantages of blacklist/whitelist technology are simplicity and efficiency. They can also be introduced into the Grid services access control area, to be used to establish a simple and effective authorization mechanism. A blacklist can be a collection of requestors that are never allowed to access a Grid service. If the authorization mechanism detects the requestor on the blacklist, it will always deny the request immediately. A whitelist is a collection of requestors that are allowed to access a Grid service. When the authorization mechanism detects the requestor on the whitelist, it will give the access permission to the requestor immediately. Based on this idea, we designed and implemented a prototype BlackListPDP and WhiteListPDP under the GT4 authorization framework. The Blacklist/whitelist-based authorization structure is shown in Fig. 6.

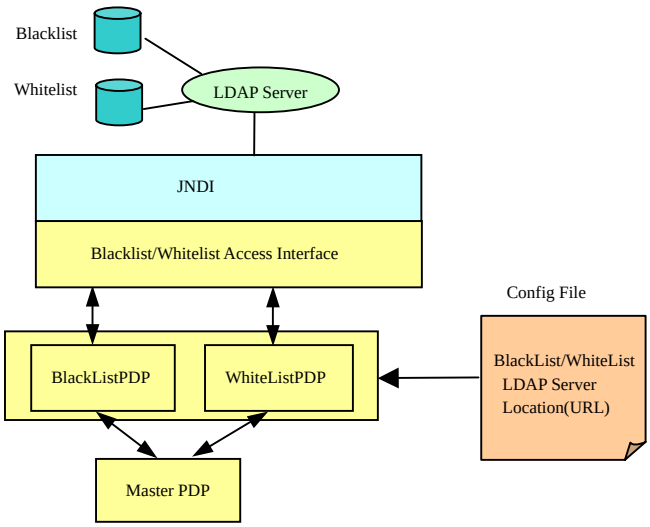

Fig. 6 Blacklist/Whitelist-based Authorization Structure

The BlackListPDP and the WhiteListPDP are inherited from the PDP abstraction introduced in Section 3.2.1. The implementation of these two PDPs has two layers: the functional layer and the implementation layer, shown in Fig. 7.

<table>
<tr><td>Functional Layer</td><td>Blacklist/Whitelist Access Interface</td></tr>
<tr><td rowspan="2">Implementation Layer</td><td>Java Naming and Directory Interface(JNDI)</td></tr>
<tr><td>Implementations (LDAP, Handle,…)</td></tr>
</table>

Fig. 7 Blacklist and Whitelist Implementation Layers

The blacklist/whitelist access interface that provides operations is defined at an abstract level and is independent of any implementation technology. We have defined a BlackList class and a WhiteList class that provide an interface for blacklist/whitelist member testing that is public boolean isMember( ).The implementation layer provides blacklist/whitelist backend. JNDI is a well-formed naming and directory integration platform. Through JNDI, we can easily use several naming and directory services to implement the blacklist/whitelist. In our prototype we use an LDAP server to store and manage the blacklist and the whitelist. The URL of the LDAP server is passed to the BlackListPDP and WhiteListPDP through a configuration file.

The blacklist and whitelist are composed of attributes of requestors, such as DN (Distinguished Name, which can be abstracted from the requestor's X.509 certificate), name, and email address. Since the security of the Grid system is based on PKI and the X.509 certificate is widely used in user authentication and authorization,[14] we chose the DN as the identity attribute of the entities in the blacklist and whitelist. A

Grid system can use other attributes such as username and group membership as the identity attributes. This can be achieved by establishing a blacklist/whitelist PIP, which obtains these identity attributes by querying an outside attribute authority using the requestor's DN, and then provides the identity attributes to the BlackListPDP or WhiteListPDP. This will provides more flexibility for users in different application environments.

The blacklist/whitelist-based authorization can be used together with other authorization mechanisms to make an efficient and rigorous authorization system. The BlackListPDP or the WhiteListPDP can form a PDP chain with other PDPs. The Master PDP will first call the BlackListPDP or the WhiteListPDP; if the requestor is not found here, the Master PDP will call other PDPs to do further decision making.

5. Conclusion

We have built a flexible multipolicy authorization framework for GT4. The framework is based on the XACML and SAML specifications. The blacklist/whitelist authorization system established under the GT4 authorization framework can provide a simple and efficient method for Grid service access control. Also, this work illustrates that the GT4 authorization framework is open, scalable, and flexible. The framework is still under development. We expect to provide a more stable version in GT4.2 which will be published later this year.

Acknowledgments

Work on GT4 GSI has been funded in part by NSF, by IBM, and by the U.S.

Department of Energy under Contract W-31-109-Eng-38. A number of individuals have made contributions to GT4 authorization framework: Von Welch, Takuya Mori, Karl Czajkowski, Jarek Gawor, Carl Kesselman, Sam Meder, Laura Pearlman, and Steven Tuecke.

References

[1] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001.

[2] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids**.** In Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.

[3] Von Welch, Frank Siebenlist, Ian Foster, John Brresnahan, Karl Czajkowski, Jarek Gawor, Carl Kesselman, Sam Meder, Laura Pearlman, and Steven Tuedke, Security for Grid Services. Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), June 2003.

[4] I. Foster and C. Kesselman. The Globus Project: A Status Report**.** Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop, pp. 4-18, 1998.

[5] Ian Foster, Carl Kesselman, Laura Pearlman, Steven Tuecke, and Von Welch. The Community Authorization Service: Status and Future. In Proc. Computing in High Energy Physics 03 (CHEP '03), 2003.

[6] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid

Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[7] The Globus Security Team, Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective, 2005.9

[8] OASIS, extensible Access Control Markup Language (XACML), V2.0, 2005.1

[9] Von Welch, Tom Barton, Kate Keahey, and Frank Siebenlist. Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration. In 4th Annual PKI R&D Workshop, April 2005.

[10] EU DataGrid, VOMS Architecture v1.1. 2003. http://gridauth.infn.it/docs/VOMS-v1_1.pdf.

[11] D. W. Chadwick, and A. Otenko, The PERMIS X.509 Role Based Privilege Management Infrastructure. 7th ACM Symposium on Access Control Models and Technologies, 2002.

[12] Tom Barton, Jim Basney, Tim Freeman, Tom Scavo, Frank Siebenlist, Von Welch, Rachana Ananthakrishnan, Bill Baker, and Kate Keahey, Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, Gridshib, and MyProxy. In 5th Annual PKI R&D Workshop (To appear), October 2005.

[13] Faronics White paper, Blacklist Versus Whitelist Software Solutions, 2005.8

[14] V. Welch, I. Foster, C. Kesselman, O. Mulmo, Laura Pearlman, Frank Siebenlist, Steven Tuecke, and Von Welch, X.509 Proxy Certificates for Dynamic Delegation**.** 3rd Annual PKI R&D Workshop, 2004.