Fine-Grain Authorization Policies in the GRID: Design and Implementation

K. Keahey
Argonne National Lab, Argonne, IL, USA

V. Welch
University of Chicago, Chicago, IL, USA

S. Lang
Argonne National Lab, Argonne, IL, USA

B. Liu
University of Houston, Houston, TX, USA

S. Meder
University of Chicago, Chicago, IL, USA

Abstract

In this paper we describe our work on enabling fine-grain authorization for resource usage and management. We address the need of Virtual Organizations (VOs) to enforce their own polices in addition to those of the resource owners, both in regard to resource consumption and job management. To implement this design we propose changes and extensions to the Globus Toolkit's version 2 (GT2) resource management mechanism. We describe the prototype and the policy language that we designed to express fine-grain policies. We then analyze our solution and describe plans for future work.

1. Introduction

Virtual Organizations (VOs) [1] have become a common way to structure collaborations where both participants and resources may be distributed not only geographically, but also across different organizational domains. A traditional mode of operation requires users to establish direct relationships (ie., in the form of user accounts) with resources they wished to use but didn't own. In a Grid environment, where both the resource pool and the user pool are large and change dynamically, this model becomes unmanageably complex. We therefore observe a trend towards making VO credentials, used in conjunction with resource provider policies, the basis of sharing in Grids. In the VO model, resource providers typically outsource some subset of their policy administration to the VO. This allows the VO to coordinate policy across resources in different domains to form a consistent policy environment in which its participants can operate. Such environment requires mechanisms for the *specification and enforcement of VO-wide policies* allowing the VO to enforce VO-specific policies on tasks and resources owned by VO participants.

Another trend developing as the Grid potential becomes realized is *the need to express and enforce fine-grain policies* on the usage of resources. These can no longer be expressed by simple access control as the manager want to specify exactly what fractions or configurations of resource may be used by a given entity. In addition, while some VOs are focused on sharing of hardware resources (e.g. CPUs and storage), for others the primary motivation is to coordinate sharing of application services [2] requiring access to both software and hardware. In these cases the VO members should not be running arbitrary code, but only applications sanctioned by VO policy. This policy may also be dynamic, adapting over time depending on factors such as current resource utilization, a member's role in the VO, an active demo for a funding agency that should have priority, etc.

In this paper, we answer the requirements posed by these two trends. We present a design for service and resource management that enables a VO and resource managers to specify fine-grain service and resource usage policies using VO credentials and allows resources to enforce those policies. We implement our design as extensions to the Globus Toolkit version 2 (GT2) resource management mechanism [3]. We then consider policy enforcement in the context of two types of policy targets: application services, and traditional computing resources. A prototype of this implementation, combined with the Akenti authorization system [4], was demonstrated at the SC02 conference and is currently being adopted by the National Fusion Collaboratory [2].

This paper is organized as follows. In section 2, we present a use case scenario and concrete requirements guiding our design. In section 3 we define our problem.

We follow this by a discussion of the capabilities of the Globus Toolkit's resource management (GRAM) [5] mechanism and describe extensions needed to GRAM to support our architecture. Finally, we analyze our solution and conclude the paper.

2. Use Case Scenario and Requirements

In a typical VO scenario, a resource provider has reached an agreement with a VO to allow the VO to use some resource allocation. The resource providers think of the allocation in a coarse-grained manner: they are concerned about how many resources the VO can use as a whole, but they are not concerned about how allocation is used inside the VO.

The finer-grained specification of resource usage among the VO participants is the responsibility of the VO. For example, the VO has two primary classifications of its members:

- One group developing, installing and debugging the application services used by the VO to perform their scientific computation. This group may need to run many types of processes (e.g. compilers, debuggers, applications services) in order to debug and deploy the VO application services, but should be consuming small amounts of traditional computing resources (e.g. CPU, disk and bandwidth) in doing so.
- The second group performs analysis using the application services. This group may need the ability to consume large amounts of resources in order to run simulations related to their research.

Thus, the VO may wish to specify finer-grain policies that certain users may use more or less resources than others. These policies may be dynamic and change over time as critical deadlines approach.

In addition to policy on the resource utilization, the VO wishes to be able to manage jobs running on VO resources. For example, users often have long-running computational jobs using VO resources and the VO often has short-notice high-priority jobs that require immediate access to resources. This requires suspending existing jobs to free up resources; something that normally only the user that submitted the job has the right to do. Since going through the user who submitted the original job may not always be an option, the VO wants to give a group of it's members the ability to manage any jobs using VO resources so they can instantiate high-priority jobs on short notice.

Supporting this scenario places several requirements on the authorization policy system:

1. *Combining policies from different sources.* In outsourcing a portion the policy administration to the VO, the policy enforcement mechanism on the resource needs to be able to combine policies from two different sources: the resource owner and the VO.
2. *Fine-grain control of how resources are used.* For the VO to express the differences between how its user groups are allowed to use resources, the VO needs to be able to express policies regarding a variety of aspects of resource usage, not just grant access.
3. *VO-wide management of jobs and resource allocations*. The VO wants to be able to treat jobs as resources themselves that can be managed. This poses a particular challenge since jobs are dynamic, so static methods of policy management are not effective. Users may also start jobs that shouldn't be under the domain of the VO - e.g. a user may have allocations on a resource besides through the VO and jobs invoked under this alternate allocation should not be subject to VO policy.
4. *Fine-grain, dynamic enforcement mechanisms.* In order to support any policies, there must be enforcement mechanisms capable of supporting them. Most resources today are capable of policy enforcement at the user level, that is, all jobs run by a given user will have the same policy applied to them. And these mechanisms are typically statically configured through file permissions, quota and the like. Our scenario brings out the requirement enforcement mechanism needs to handle dynamic, fine-grain policies.

3. Interaction Model

To support the use case described in the previous section, we need to provide resource management mechanisms that allow the specification and consistent enforcement of authorization and usage policies that come from both the VO and the resource owner. In addition to allowing the VO to specify policies on standard computational resources, like processor time and storage, we need to allow the VO to specify policies on application services that it deploys as well as long-running computational jobs instantiated by VO members.

In our work we will assume the following interaction model:

1. A user submitting a request, composed of the job's description, initiates a job. The request is accompanied by the user's Grid credentials, which may include the user's personal credentials as well as VO-issued credentials.
2. This request is evaluated against both local and VO policies by different policy evaluation points (PEPs), capable of interpreting the VO and the resource management policy respectively, located in the resource management facilities.
3. If the request is authorized by both PEPs, it is mapped to a set of local resource credentials (e.g. a Unix user account). Policy enforcement is carried out by local enforcement mechanisms operating based on local credentials.
4. During the job execution, a VO user may make management requests to the job (e.g. request information, suspend or resume a job, cancel a job).

4. Grid Resource Management in GT2

Grids are the collection of middleware needed to support VOs. The Globus Toolkit® is an implementation of a Grid infrastructure. It provides mechanisms for security, data management and movement, resource monitoring and discovery (MDS) and resource acquisition and management. In this paper we are focusing on the functionality of resource acquisition and management, which is implemented by the GRAM (Grid Resource Acquisition and Management) system [5].

The GRAM system has two major software components: the Gatekeeper and the Job Manager. The Gatekeeper is responsible for translating Grid credentials to local credentials (e.g. mapping the user to a local account based on their Grid credentials) and creating a Job Manager Instance to handle the specific job invocation request. The Job Manager Instance (JMI) is a Grid service which instantiates and then provides for the ability to manage a job. Figure 1 shows the interaction of these elements; in this section we explain their roles and limitations.

4.1. Gatekeeper

The Gatekeeper is responsible for authenticating the requesting Grid user, authorizing their job invocation request and determining the account in which their job should be run. Authentication, done using the Globus Toolkit's Grid Security Infrastructure [13], verifies the validity of the presented Grid credentials, the user's possession of those credentials and the user's Grid identity as indicated by those credentials. Authorization is based on the user's Grid identity and a policy contained in a configuration file, the *grid-mapfile*, which serves as an access control list. Mapping from the Grid identity to a local account is also done with the policy in the grid-mapfile, effectively translating the user's Grid credential into a local user credential. Finally, the Gatekeeper starts up a Job Manage Instance (JMI), executing with the user's local credential. This mode of operation requires the user to have an account on the resource and implements enforcement by privileges of the account.

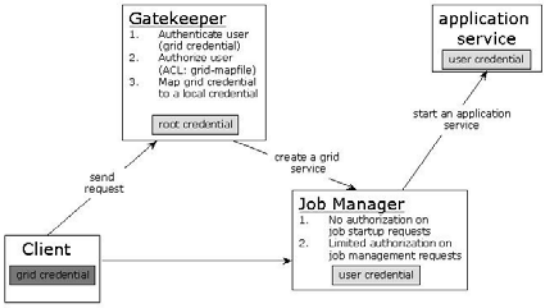

Figure 1: Interaction of the main components of GRAM

4.2 Job Manager Instance (JMI)

The JMI parses the user's request, including the job description, and interfaces with the resource's job control system (e.g. LSF, PBS) to initiate the user's job. During the job's execution the JMI monitors its progress and handles job management requests (e.g. suspend, stop, query, etc.) from the user. As the JMI is run under the user's local credential, as defined by the user's account, the operating system, and local job control system are able to enforce local policy on the JMI and user job by the policy tied to that account.

The JMI does no authorization on job startup since the Gatekeeper has already done so. However, once the job has been started, the JMI accepts, authenticates and authorizes management requests on the job. In GT2, the authorization policy on these management requests is static and simple: the Grid identity of the user making the request must match the Grid identity of the user who initiated the job.

4.3. GRAM Shortcomings

The current GRAM architecture has a number of shortcomings when matched up with the requirements we laid out in Section 2:

1. Authorization of user job startup is coarse-grained. It is based solely on whether a user has an account on a resource.
2. Authorization on job management is coarse-grained and static. Only the user who initiated a job is allowed to manage it.
3. Enforcement is implemented chiefly through the medium of privileges tied to a statically configured local account (JMI runs under local user credential) and is therefore useless for enforcing fine-grained policy or dynamic policy coming from sources external to the resource (such as a VO).
4. Local enforcement depends on the rights attached to the user's account, not the rights presented by the user with a specific request; in other words, the enforcement vehicle is largely accidental.
5. A local account must exist for a user; as described in the introduction, this creates an undue burden on system administrators and users alike. This burden prevents wide adoption of the network services model in large and dynamically changing communities.

These problems can, and have been, in some measure alleviated by clever setup. For example, the impact of (4) can be alleviated by mapping a grid identity to several different local accounts with different capabilities. (5) is often coped with by working with "shared accounts" (which however introduces many security, audit, accounting and other problems) or by providing a limited implementation of dynamic accounts [6].

5. Authorization and Enforcement Extensions to GRAM

In this section we describe extensions to the GT2 Grid Resource Acquisition and Management (GRAM) that address the shortcomings described above.

We extended the GRAM design to allow authorization callouts, evaluating the user's job invocation and management requests in the context of policies defined by the resource owner and VO. Out changes to GRAM, prototyped using GT2, are illustrated in figure 2.

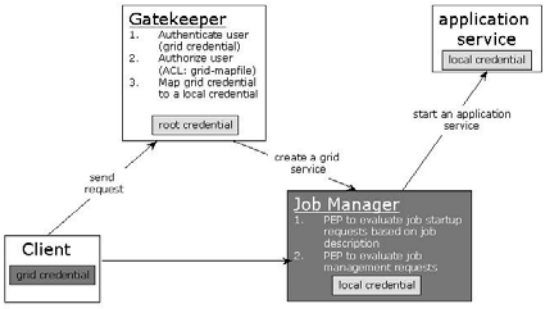

Figure 2: Changes to GRAM; the changed component (the Job Manager) has been highlighted in gray.

In our prototype we experimented with policies written in plain text files on the resource. These files included both local resource and VO policies (in a real system the VO policies would be carried in the VO credentials). This work has recently been tested with the Akenti [4] system representing the same policies as described here, and is being adopted by the National Fusion Collaboratory [2]. In order to show generality of our approach, we are also experimenting with the Community Authorization Service (CAS) [7]. Both of these systems allow for multiple policies sources, but have significant differences, both in terms of architecture and programming APIs.

5.1. Policy Language

GRAM allows users to start and manage jobs by submitting requests composed of an *action,* (e.g. initiate, cancel, provide status, change priority, etc.), and, in the case of job initiation, a job description. The job description is formulated in terms of attributes using the Resource Specification Language (RSL) [3]. RSL consists of attribute value pairs specifying job parameters referring to executable description (executable name, directory where it is located, etc.), and resource requirements (number of CPUs to be used, maximum/minimum allowable memory, maximum time a job is allowed to run, etc.).

We have designed a simple policy language that allows for policy specification in terms of RSL. The policy assumes that unless a specific stipulation has been made, an action will not be allowed. It is expressed as a set of assertions where a user, or a group of users, is related to a set of assertions. The rules have the form of user (or group) identity separated by a colon from a set of action-based assertions that follow the RSL syntax.

In order to express the rules we extended the RSL set of attributes with the addition of the following:

- *Action.* The action attribute action represents what the user wants to do with the job, and currently can take on values of “start”, “cancel”, “information”, or “signal”, where signal describes a variety of job management actions such as changing priority.
- *Jobowner*. The jobowner attribute denotes the job initiator and can take on values of the distinguished name of a job initiator’s grid credential. It is used mainly to express VO-wide management policy.
- *Jobtag*. The jobtag attribute has been introduced in order to enable the specification of VO-wide job management policies. A jobtag indicates the job membership in a group of jobs for which policy can be defined. For example, a set of users with an administrative role in the VO can be granted the right to manage all jobs in a particular group. A policy may require a VO user to submit a job with a specific jobtag, hence placing it into a group that is manageable by another user (or group of users). At present jobtags are statically defined by a policy administrator.

We also added the following values to RSL:

- “NULL” to denote a non-empty value
- “self” to allow expression of the job initiator's identity in a policy.

These extensions allow following types of assertions to be expressed in policy:

- The job request is permitted to contain a particular attribute a particular value or set of values. This allows, for example, the maximum number of processors used to be limited or to restrict the name of the executable to a specified set. Multiple assertions can be made about the same attribute.
- The job request is required to contain a particular attribute, possibly with a particular value or set of values. For example, the job request must specify a jobtag attribute to allow its management by a VO-defined group of administrators.
- The job request is required not to contain a particular attribute. Either at all or just with a particular value or set of values. For example, the job request must not specify a particular queue, which is reserved for high-priority certain users.

Our extensions allows a policy to limit not only the usage of traditional computational resources, but to dictate the executables they are allowed to invoke, allowing a VO to limit the way in which they can consume resources. Further, by introducing the notion of a jobtag we are able to express policies allowing users to manage jobs. The example in figure 3 illustrates how policy may be expressed.

The fist statement in the policy specifies a requirement for a group of users whose Grid identities start with the string " /O=Grid/O=Globus/OU=mcs.anl.gov". The requirement is that for job invocations (where the action is "start"), the job description must contain a jobtag attribute with some value. This allows us to later write management policies referring to a specific jobtag.

&/O=Grid/O=Globus/OU=mcs.anl.gov:
(action = start)(jobtag != NULL)

/O=Grid/O=Globus/OU=mcs.anl.gov/CN= Bo Liu:
&(action = start)(executable = test1)(directory = /sandbox/test)(jobtag = ADS)(count<4)
&(action = start)(executable = test2)(directory = /sandbox/test)(jobtag = NFC)(count<4)

/O=Grid/O=GlobusOU=mcs.anl.gov/CN= KateKeahey:
&(action = start)(executable = TRANSP)(directory = /sandbox/test)(jobtag = NFC)
&(action=cancel)(jobtag=NFC)

Figure 3: Simple VO-wide policy for job management

The second statement in the policy refers to a specific user, Bo Liu, and states that she can only start jobs using the "test1" and "test2" executables. The rules also place constraints on the directory from which the executable can be taken and the jobtag they can be started with. In addition, a constraint is placed on the number of processors Bo Liu can use (count < 4).

The third statement in the policy gives user Kate Keahey the right to start jobs using the "TRANSP" executable from a specific directory and with a specific jobtag. It also gives her the right to cancel all the jobs with jobtag "NFC"; for example, jobs based on the executable "test1" started by Bo Liu.

5.2. Enforcing Policies in GRAM

We enforce our policies in GRAM by creating a policy evaluation point (PEP) controlling all external access to a resource via GRAM; an action is authorized depending on decision yielded by the PEP. Policy can be enforced in GRAM at multiple PEPs corresponding to different decision domains; for example a PEP placed in the Gatekeeper can allow or disallow access based on the user's Grid identity. Since our work focuses on job and resource management we established a PEP in the Job Manager (JM). The JM parses user job descriptions and can therefore evaluate policy that depends on the nature of the job request in addition to the user's identity.

Specifically, our additions consisted of the following:

- Designing an authorization callout API to integrate the PEP with the JM. The callout passes to the PEP authorization module the relevant information, such as: the credential of the user requesting a remote job, the credential of the user who originally started the job, the action to be performed (such as start or cancel a job), a unique job identifier, and the job description expressed in RSL. The PEP responds through the callout API with either success or an appropriate authorization error. This call is made whenever an action needs to be authorized; that is before creating a job manager request, and before calls to cancel, query, and signal a running job.
- Policy-based authorization for job management. As discussed in section 4, each job management request other than job startup is currently authorized by GRAM so that only the user that started a job is allowed to manage it. We modified the authorization in GRAM to enable Grid users other than the job initiator to manage the job based on policy with decisions rendered through the authorization callout API. In addition to changes to the authorization model, this also required extensions to the GRAM client allowing the client to process other identities than that of the client (specifically, allowing it to recognize the identity of the job originator).
- RSL parameters. We extended RSL to add the "jobtag" parameter allowing the user to submit a job to a specific job management group.
- Errors. We further extended the GRAM protocol to return authorization errors describing reasons for authorization denial as well as authorization system failures.

In order to provide for easy integration of third party authorization solutions, the callout API provides facilities for runtime configurable callouts. Callouts can be configured either through a configuration file or an API call. Configuration consists of specifying an abstract callout name, the path to the dynamic library that implements the callout and the symbol for the callout in the library. Callouts are invoked through runtime loading of dynamic libraries using GNU Libtool's dlopen-like portability library. Arguments to the callout are passed using the C variable argument list facility.

The insertion of callout points into JM required defining a GRAM authorization callout type, i.e. a abstract callout type, the exact arguments passed to the callout and a set of errors the callout may return. These callout points are configured by parsing a global configuration file.

6. Analysis

Our solution overcame some of the shortcomings outlined in section 4.3. However our approach has a number of problems and outstanding issues that we discuss in this section.

6.1. Gateway Enforcement Model

A weakness of the gateway approach is that once a gateway authorizes an action (for example a job execution); it is no longer involved in the continuous enforcement of the policy. GRAM maps incoming requests to static local accounts to perform this continuous policy enforcement.

This has two consequences: (1) the local policy enforcement depends on the privileges tied the account that the user maps to on the local system rather than to the credential with which the request was made, and (2) GRAM's abilities for continuous policy enforcement are limited by local capabilities for policy enforcement.

The first limitation could be to some extent dealt with by using dynamic accounts. Dynamic Accounts are accounts created and configured on the fly by a resource management facility. This enables the resource management system to run jobs on a system for users that do not have an account on that system, and it also enables account configuration relevant to policies for a particular resource management request as opposed to a static user's configuration. To some extent a dynamic account can be also used as a sandbox on the user's rights (by modifying user's group membership to control file system access for example). On the other hand, although work has been done to support fine-grain policy for file access [8], in general accounts allow the user to modify only very few configuration parameters, and hence the enforcement implemented in an account is coarse-grained. For this reason, dynamic accounts may need to be supplemented by sandboxing.

A sandbox is an environment that imposes restrictions on resource usage [9]. Sandboxing represents a strong enforcement solution, having the resource operating system act as the policy evaluation and enforcement modules and is largely complementary to the gateway approach. However, while they provide a solution with relatively high degree of security, they are hard to implement portably and may introduce a performance penalty

.

6.2. Job Manager Trust Model

In the GRAM architecture, the job manager runs with the user's local credentials; this makes the job manager a less than ideal vehicle for policy enforcement. The reasons for that are twofold. First, from the security perspective this makes it a poor choice for a policy enforcement point since it is vulnerable to tampering by the user that could result in changed in policy enforcement. Secondly, this effectively limits enforcement potential for VO-wide job management. For example, a user managing a job may cancel a job started by somebody else (by virtue of the fact that the job manager is running with the job initiator's local credential), but they may not apply their higher resource rights to, for example, raise the job's priority.

One possible solution to this problem in the context of GRAM architecture would be to locate the policy enforcement point in the gatekeeper. However, this would increase the vulnerability of the system by placing more complex code into the trusted component of the system, increasing chances for logic errors, buffer overflows, etc.

Another possibility would be for policy enforcement to be done by trusted services like the local operating system. As discussed earlier, this is difficult today because most operating systems do not have the support for fine-grain policies that we require. Investigation into sandboxing techniques remains an open research issue.

6.3 Policy Language

Our implementation currently expresses policy in terms of the same resource specification language (RSL) that GRAM uses to describe jobs. While this allows for easy comparison of a job description with a policy, it is not a standard policy language. Policy administrators are not familiar with RSL, and our initial experiences show that expressing policies in these terms is not natural to this community. This difficulty is compounded by the fact that the syntax is not be supported by standard policy tools. We are therefore investigating existing policy languages as a replacement to our RSL-based scheme. With the merging of Grid technologies and Web Service-based technologies in OGSA[10], languages based on XML, such as XACML [11] and XrML [12], are being scrutinized by the Grid security community in general and are viable candidates.

6. Conclusions and Future Work

We have described the design and implementation of an authorization system allowing for enforcement of fine-grained policies and VO-wide management of remote jobs. To implement this design we proposed changes to the Globus Toolkit GRAM design and designed a policy language suitable for our needs. We are planning to use the same mechanism to provide pluggable authorization in other components of the Globus Toolkit.

While our work solves some of the problems with GRAM, it also leaves some open questions, mainly in the area of enforcement, where sandboxing and dynamic account management remain open questions. Since our work began, a new version of GRAM has

been releases as part of version 3 of the Globus Toolkit (GT3). The new GRAM design, described in [13], offers some enhancements that we see benefiting our work. For example, the job description is available to a trusted service as part of job creation, which allows it to configure the local account, and creates potential for better integration with dynamic accounts.

Acknowledgements

We are pleased to acknowledge contributions to this work by Mary Thompson of LBNL. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under contracts W-31-109-Eng-38, DE-AC03-76SF0098, DE-FC03-99ER25397 and No. 53-4540-0080.

Bibliography

1. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of High Performance Computing Applications, 2001. **15**(3): p. 200-222.
2. Keahey, K., T. Fredian, Q. Peng, D.P. Schissel, M. Thompson, I. Foster, M. Greenwald, and D. McCune, *Computational Grids in Action: the National Fusion Collaboratory.* Future Generation Computing Systems (to appear), October 2002. **18**(8): p. 1005-1015.
3. Czajkowski, K., I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, *A Resource Management Architecture for Metacomputing Systems*, in *4th Workshop on Job Scheduling Strategies for Parallel Processing*. 1998, Springer-Verlag. p. 62-82.
4. Mary Thompson, W.J., Srilekha Mudumbai, Gary Hoo, Keith Jackson, Abdelilah Essiari, *Certificate-based Access Control for Widely Distributed Resources*, in *Proc. 8th Usenix Security Symposium*. 1999.
5. Butler, R., D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, *Design and Deployment of a National-Scale Authentication Infrastructure.* IEEE Computer, 2000. **33**(12): p. 60-66.
6. *dynamic accounts.* http://www.gridpp.ac.uk/gridmapdir/.
7. L. Pearlman, V.W., I. Foster, C. Kesselman, S. Tuecke. *A Community Authorization Service for Group Collaboration*. in *submitted to IEEE Worksop on Policies for Distributed Systems and Networks*. 2002.
8. Lorch M. and K. D. *Supporting Secure Ad-hoc User Collaboration in Grid Environments*. in *Proceedings of the 3rd Int. Workshop on Grid Computing - Grid 2002, Baltimore, MD, USA*. 2002.
9. Chang, F., A. Itzkovitz, and V. Karamacheti, *User-level Resource-constrained Sandboxing.* Proceedings of the USENIX Windows Systems Symposium (previously USENIX-NT), 2000.
10. Foster, I., C. Kesselman, J. Nick, and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. 2002: Open Grid Service Infrastructure WG, Global Grid Forum,.
11. *OASIS eXtensible Access Control Markup Language (XACML) Committee Specification 1.0 (Revision 1).* http://www.oasis-open.org/committees/xacml/docs/s-xacml-specification-1.0-1.doc, 2002.
12. *XRML.* http://www.xrml.org/get_XrML.asp.
13. Welch, V., F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, *GSI3: Security for Grid Services (Draft).* Submitted to HPDC 2003.