The PRIMA Grid Authorization System

Markus Lorch and Dennis Kafura
{mlorch@vt.edu, kafura@cs.vt.edu}

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

Abstract

PRIMA, a system for PRIvilege Management and Authorization, provides enhanced grid security services. The requirements for these services are derived from usage scenarios and supported by a survey of grid users. The requirements for added flexibility, increased expressiveness, and more precise enforcement are met by a combination of three mechanisms: (1) use of secure, fine-grained privileges representing externalized access rights for grid resources that can be freely created, shared, and employed by grid users; (2) a dynamic policy generated for each request combining the request's user-provided privileges with the resource's access control policy; and (3) dynamic execution environments specially provisioned for each request that are enforced by the resource's native operating system and which support legacy applications. PRIMA has been implemented as an extension of the Globus Toolkit grid middleware.

Key Words

Grid Security, Privilege Management, Authorization, Enforcement, Access Control

Abbreviations

API	- Application Programming Interface
CAS	- Community Authorization Service
GSS	- Generic Security Services
PDP	- Policy Decision Point
PEP	- Policy Enforcement Point
PRIMA	- A system for Privilege Management and Authorization
VO	- Virtual Organization
VOMS	- Virtual Organization Membership Service
XACML	- The eXtensible Access Control Markup Language

1 Introduction

Securing the current and future generations of computational grid infrastructures requires fundamental advances in security models and mechanisms. These advances must overcome usability and scalability limitations caused by confining authority-granting to relatively few system administrators, imposing trust establishment procedures that cater only to large organizations, and restricting the granularity of security controls to a level that interferes with least-privilege access.

This paper will discuss improvements to the security mechanisms of computational grid infrastructures based on a new security model for PRIvilege Management and Authorization (PRIMA). PRIMA empowers users to act authoritatively for their own resources, supports virtual organizations (VOs) [1] based on small, dynamic working groups for which the ability to establish transient collaboration is a key requirement, and provides fine-grained expression and enforcement of access rights. The mechanisms introduced allow system administrators to retain final authority over the use of resources. The PRIMA implementation interfaces with and enhances the security mechanisms of the Globus Toolkit [2][3].

To illustrate the need for improved grid security two usage scenarios for advanced grid environments are described from which a set of security requirements is derived. The scenarios are based on our experience from previous work [4][5], interactions with other grid researchers and grid resource providers [6][7], and on the results of a grid community survey summarized below.

Scenario S1 – Ad-hoc Collaboration: *Bob is a university researcher who has developed a protocol emulator executed on a dedicated cluster resource for which he is authoritative. He would like to grant Joan, a researcher in industry, temporary access to his emulator and the cluster resource enabling Joan to evaluate a new protocol she has developed. Given the limited nature of the collaboration, neither Bob nor Joan wishes to involve system administrators or perform low level modifications of operating systems specific details.*

Scenario S2 – Multi-Project User: *Sam takes part in a number of distinct projects that utilize shared grid resources. He must ensure that his utilization of grid resources is accounted to the correct project. Furthermore, he requires grid mechanisms to protect his results and parameters from accidental modification or deletion by his colleagues. These mechanisms must not constrain his ability to share these data individually or in bulk with his colleagues. He frequently executes experimental code produced by him and by external collaborators besides his production code. Sam must ensure that the experimental code does not interfere with the execution of the production code and would like to be able to define exactly what resource objects an experimental code can access.*

These scenarios are consistent with two findings of a survey conducted among members of the grid community to understand the needs of grid user and grid application developers, provide information on typical modes-of-use, and elicit requirements for future grid security systems. The survey was announced and distributed at the sixth meeting of the Global Grid Forum in fall of 2002. Most (77%) of the 39 respondents classified themselves as grid researchers and developers. The detailed results of this survey are documented in [8].

First, the majority of today's grid communities tend to be relatively small in size and use a focused set of the total available resources. The majority (60%) of the respondents are in collaborations of 1 to 25 members. Most respondents (71%) that are member of grid communities answered that they make use of less than 10% of the total number of community resources. These results point to the need for security mechanisms catering to small collaborative efforts using a targeted set of resources as described in scenario S1.

Second, the plurality of grid users decide for themselves with whom they want to collaborate and what rights should be allocated to their collaborators. In more than half (54%) of the cases of ad-hoc communities, privileges are assigned through a community administrator, which may be evidence of the lack of support for efficient management of privileges in ad-hoc grid collaboration scenarios. The plurality of respondents (41%) decide for themselves with whom they share resources, of which 37.5% have to contact an administrator in order to grant the corresponding access rights. These numbers underscore the need for grid users to be able to manage access to their resources directly as described in scenario S1 and S2.

The two usage scenarios reveal a number of security requirements for grid systems. The most important of these requirements are:

Fully distributed mechanisms: Scenario S1 describes an ad-hoc collaboration that cannot rely on group infrastructure such as servers that manage group membership and access permissions. The security mechanisms used to establish and maintain short lived, ad-hoc collaborations must be fully distributed and should not require group-specific infrastructure components.

Fine-grained access rights: An access right is "fine-grained" if it denotes a specific right of a given user to a given object. Access rights are not fine-grained if they require a right to be granted to all members in a group of users or if the right conveys access to all objects in a set of objects. Both scenarios require fine-grain rights. For example, Bob in scenario S1 must be able to grant Joan only limited access to the compute cluster. In scenario 2, Sam must be able to authorize access to specific data files and programs.

Direct delegation of authorization: Scenario S1 points out the need for users and other authoritative parties to directly delegate access to other grid entities (users and processes). Direct delegation is especially powerful when combined with fine-grained access rights as the minimum access rights required for a specific task or purpose can be delegated to collaborators. The ability to delegate access rights directly is also needed to provide for the necessary scalability of grid security solutions. If delegation is only supported via indirect means, such as the need to involve a system administrator that adapts resource access policies to reflect the delegation, the overhead required to delegate authorization creates barriers to the scalability of such a system.

Selective use of access rights: Both, scenario S1 and scenario S2 reveal the need for grid entities to be able to select which of their access rights should be used to authorize a specific access request. While this does not guarantee that access will be requested with only the least amount of privileges, it does empower the user to reduce the set of privileges a specific request will be served with and thus has the potential to increase overall system security (tools may be used to aid the user in the selection of a minimal set of privileges). Furthermore, the ability for an entity to select a different subset of its rights for unrelated

requests allows for the effective logical separation of service requests and associated data. Concurrent services executing on behalf of the same entity are less likely to suffer from interference if they each only have access to the resources they require.

Fine grain enforcement: To support fine grain access rights the enforcement mechanisms must be able to constrain the execution of services at an equally fine level of granularity. This requirement stems from both scenarios S1 and S2.

Support for legacy and untrusted applications: Resource owners and users require the ability to constrain the resource access of legacy applications and untrusted code to prevent accidental or malicious misuse of resources as presented in scenario S2. Legacy code is typically unable to constrain its use of resources and resources thus must provide an execution environment that will constrain these applications. New applications could be written with internal access control enforcement mechanisms such that these applications could run without underlying enforcement mechanisms in place. However, this requires the application code to be fully trusted by the resource to perform such enforcement in compliance with the applicable policies. Entities involved in ad-hoc groups only place minimal trust in (and thus require high protection from) executables received from new collaborators. In grid environments custom codes, which cannot be trusted by resource owners, are often executed on behalf of the user to render a service.

Basic security requirements, such as the support for single sign-on and mutual authentication, have been omitted from this list since these requirements are well met by existing grid systems.

The survey indicated that support for these requirements is missing in current grid systems. For example, the survey results showed that existing grid security solutions do not provide adequate services for collaborative grid communities (53%), mechanisms for a user to manage their own privileges and credentials are insufficient (59%), and real world trust relationships can not be modeled adequately (56%). One of the survey respondents stressed the need to "*push privilege management down to the individuals, which poses a challenge to the tools and plumbing*" which nicely captures the requirements motivated by the two scenarios.

PRIMA addresses the security requirements through a unique combination of three innovative approaches:

- **Privileges**: unforgeable, self-contained, fine-grained, time limited representations of access rights externalized from the underlying operating system,
- **Dynamic policies**: a request-specific access control policy formed from the combination of user-provided privileges with a resource's access control policy to form, and
- **Dynamic execution environments**: a specifically-provisioned native execution environment limiting the use of a resource to the rights conveyed by user-supplied privileges.

In combination, these methods meet the six security requirements derived from the usage scenarios. The use of secure privileges guarantees that users can use decentralized facilities to manage their access rights by attaching only selected privileges to requests, delegate authorizations by sharing privileges with other users, and express their access rights at a fine

grained level. The dynamic policy and execution environments offer fine grain enforcement through expressive policy representation and specific provisioning of the execution environment. Finally, the dynamic execution environment supports legacy applications by using native operating system enforcement mechanisms.

The remainder of this paper is organized as follows. Section 2 outlines the major PRIMA constructs (privileges, dynamic policies, and dynamic execution environments). Section 3 describes the design and implementation of the PRIMA system. Section 4 compares PRIMA to other similar systems and offers conclusions.

2 PRIMA Concepts

2.1 Overview

Figure 2-1 provides a high-level overview of the PRIMA system using numbered arrows to represent a general sequence of actions. In step 1, resource administrators configure grid resources with access policies that regulate usage of the resource. These policies can be created and managed at the grid layer using platform independent languages such as XACML [9]. Access rights of resource objects are externalized and represented to both administrators and users on the grid layer as PRIMA privileges (step 2). For example, a privilege for a file access right is abstracted from the way file access rights are stored in the file meta information by the native operating system. The PRIMA representation uses XACML constructs to encode the externalized form of the privilege. A privilege is self-contained in that its meaning is fully determined by the information contained in the privilege. Step 3 shows that privileges can be delegated among and between grid users, administrators, and other grid entities (such as grid services, proxies, agents, etc.). Grid users holding privileges can manage the use of their privileges by selecting a subset of their privileges for use with a specific access (step 4) and bundle the subset of selected privileges with the specific grid resource requests (step 5).

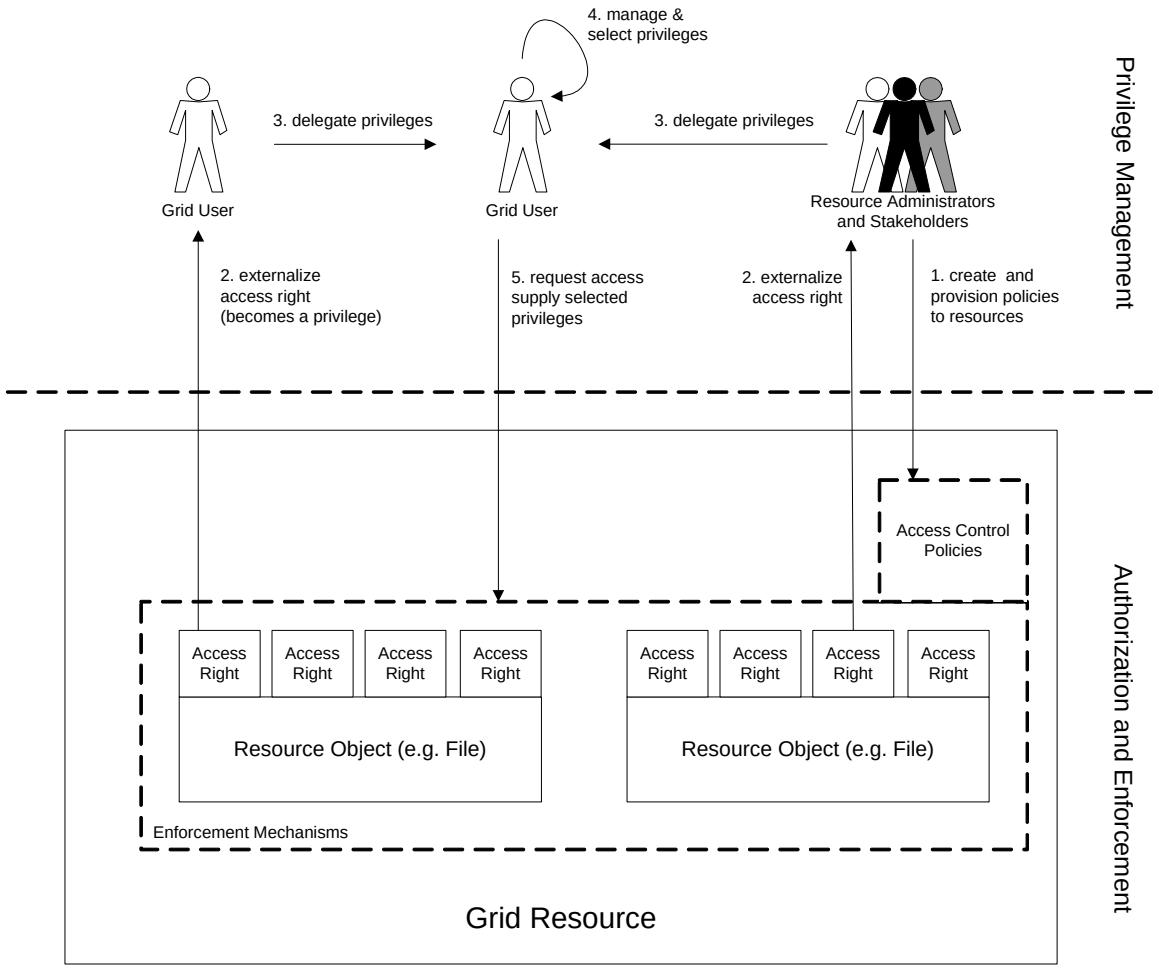

Figure 2-1 PRIMA System Overview

Because the holder of privileges can selectively provide individual privileges to grid resources when requesting access, least privilege access to resources is enabled and the user is ensured of fine-grained control over resource usage of requested services. Privileges form a key element of PRIMA. Privileges are described further in Section 2.2 Authorization decisions are made based on the resource's access policy and the privileges provided with the request. The combination of the user's privileges with the resource's security policy prior to the assessment of the user's request is termed a "dynamic policy". Dynamic policies are explained further in Section 2.3. Finally, the request is executed in an environment configured with the minimal set of fine-grained access rights required for the specific access. Enforcement via the execution environment provides for the secure execution of non-trusted legacy applications without the need to duplicate security code already present in the operating systems. This concept is explained in more detail in Section 2.4

2.2 Privileges

In PRIMA the term privilege refers to a "fully associated" and "directly applicable" access right that has been externalized from the resource's internal representation and is packaged in a container that protects it from manipulation and provides for issuer identification that cannot be repudiated.

Privileges are fully associated meaning that they explicitly specify the subjects, objects (resource) and allowed actions by the subjects on that object. The full association property means that privileges have full meaning and can be managed outside of the context of a particular resource or a particular request. Other authorization tokens such as capabilities or access control lists are not fully associated. Capabilities are explicitly bound to the action and the object but only implicitly (via possession) to a subject. Access control lists enumerate subjects and allowed actions explicitly, but the object is typically implicitly specified by linking the list as a whole with a resource object. Privileges are also distinct from authorization decisions. Authorization decisions are explicitly bound to a specific service request which in turn includes subject, object and action information. In contrast, privileges are not bound to a particular service request.

Privileges are directly applicable access rights meaning that they can be exercised without interpretation. In contrast, commonly used subject attributes like group membership or clearance level are resource-agnostic and have to be rendered against resource-specific policies to yield an applicable access right.

The access rights conveyed through privileges are externalized from their resource specific representation. Privileges are described in a platform independent format and contain meta information such as the type of privilege (used by an entity accepting privileges to select an appropriate enforcement mechanism) and the privilege issuer (needed to verify the authority of the privilege).

Privileges are secure against accidental or deliberate alteration. In PRIMA, privileges are embedded in a container as shown in Figure 2-2. The container contains issuer and holder information and provides protection against modification (e.g. through the use of digital signatures). The payload of the container is the privilege's externalized representation. The protection afforded to privileges enables resource independent, grid-layer sharing, as well as use and management of access rights outside of a protected resource environment.

PRIMA privileges have a well defined lifetime. This property also distinguishes privileges from traditional access rights that are typically not bounded in time and from authorization decisions that are implicitly bounded through their association with a single request. The lifetime property models real-world situations where entitlements expire unless they are frequently renewed. This protects systems from dormant privileges that may no longer reflect real world trust relationships (e.g. an entity may have switched organizations) and mitigates risks associated with delegation as the grounds for the delegation can be verified before a privilege is renewed. A privilege's lifetime also provides the basis for the implementation of a variety of revocation mechanisms. Due to the lifetime, the set of privileges for which revocation information may have to be provided is bounded, as expired privileges do not or no longer need to be revoked because the access rights based on privileges become void at privilege expiration. No manual management action is required on the side of the privilege issuer.

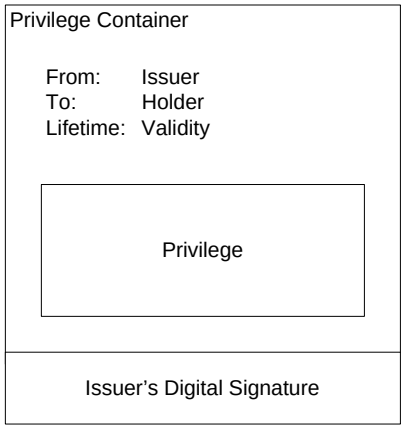

Figure 2-2 Privilege Container

PRIMA privileges are fine-grain. Fine grain is a relative quality that is interpreted with respect to the resources and applications of interest. The resources and applications of interest to PRIMA are those relevant to computational grids. In this context, the privilege "user X is allowed to read file F" is considered fine grain while the privilege "user X is allowed to exercise any right assigned to account Y" is not considered fine grain. A coarse grain privilege can, of course, be expressed in a fine grain system. In other contexts, fine grain may refer to even smaller entities than those considered in this work. For example, in [10] individual attributes in an XML document are considered fine grain.

2.3 Dynamic Policies

Figure 2-3 illustrates the authorization decision and enforcement process in PRIMA. When a subject issues a service request (step 1), the subject also provides a set of privileges. The request and the accompanying privileges are presented to a Policy Enforcement Point (PEP). For each provided privilege, the enforcement point individually checks the

- **applicability** (does the privilege apply to the local resource and the entity requesting the service),
- **validity** (is the privilege within its lifetime and is the digital signature intact), and
- **authority** (was the privilege issued by an authoritative party).

Authority to issue privileges is defined in a privilege management policy available to the Policy Decision Point (PDP). The PEP queries the PDP for authoritativeness of each privilege issuer before further considering a provided privilege (these interactions are not shown in Figure 2-3). All permissible privileges constitute the dynamic policy for the request. The enforcement point can now contact the decision point with an authorization request which includes the dynamic policy (step 2). The PDP evaluates the request against the combination of the set of applicable policies (step 3). The set of applicable policies is comprised of the resource access control policies provisioned by system administrators (see step 1 in Figure 2-1) and the dynamic policy that has been compiled by the PEP based on the set of user-supplied privileges. Once an authorization decision is reached the PDP provides a response including the decision back to the enforcement point (step 4). Conceptually two decisions are made by the PDP: (1) a relatively coarse decision corresponding to the service request that specifies if the Policy Enforcement Point (PEP) should allow access to the requested service in general; and (2) a set of instructions, termed obligations, from the PDP to the PEP on how the requested service should be confined and monitored during its execution. If the PEP cannot fulfill the obligations then it should not allow the access to proceed. The enforcement point, upon receiving a positive response from the decision point, instantiates a custom execution environment configured with the access rights as specified in the obligations, and starts and monitors the execution of the requested service in this environment (step 5). Service responses are passed to the enforcement point (step 6) and relayed to the subject (step 7).

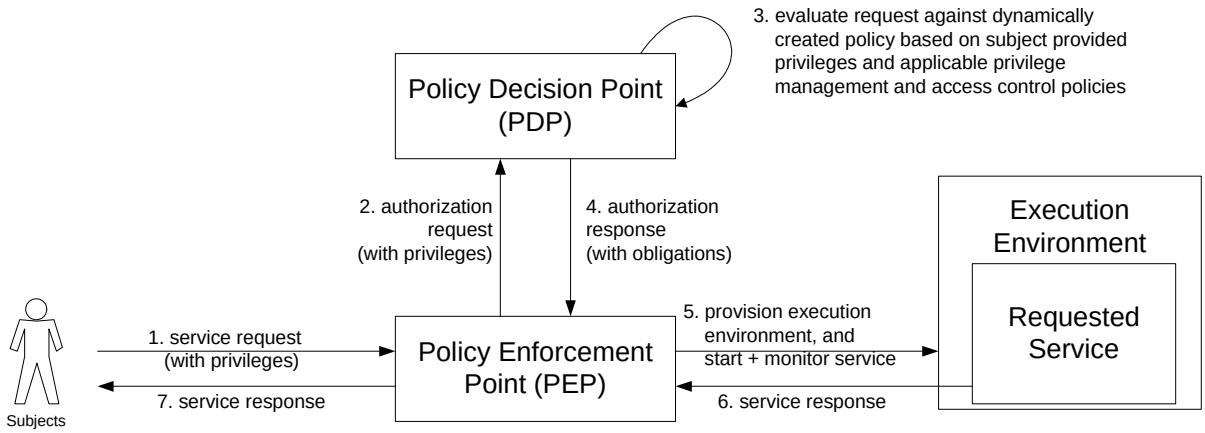

Figure 2-3 Authorization Decision and Enforcement Process

The use of obligations in authorization decisions addresses one of the more subtle issues frequently encountered when fine-grained authorization decisions are made: a mismatch in the level of detail between the authorization request and the applicable policies. Policy decision points are application independent and are unable to understand or extrapolate the implications of a broad resource request. An example is the request to instantiate a user-provided service. The applicable policies and provided privileges are likely to specify in detail what a user provided service is allowed to do. But a simple yes/no decision from the decision point cannot convey this level of detail. A positive authorization decision response thus needs to be augmented with additional decision qualifications that instruct the enforcement point how exactly the requested action should be permitted and if additional constraints should be applied. For example, the list of fine grained access rights that specifies to what extent the user provided service is allowed to access other services and resources of the hosting environment can be provided this way. In PRIMA, additional constraints to an authorization decision are referred to as obligations of an authorization decision. Obligations also enable the system to maintain state (e.g. information about previously used privileges or previous unsuccessful access attempts). The PDP can instruct the PEP to keep such state information and present it back to the PDP on subsequent access requests (e.g. in the form of additional attributes, such as environment attributes, to the authorization request). The enforcement mechanisms should only permit the requested access if the obligations can be enforced as well.

2.4 Dynamic Execution Environments

In the PRIMA enforcement model, each authorized request is executed within a specifically-provisioned execution environment. This execution environment is dynamically configured with the permissions specifically provided by the user to perform the user's request. As noted, this feature allows the user to operate in accordance with the principle of "least privilege" access. Execution environments can be implemented in a variety of ways: standard UNIX process spaces, sandboxing approaches such as [11][12][13], or hosting environments for web and grid services. These techniques enable a system built following the PRIMA model to employ native security mechanisms. The use of native mechanisms results in lower administrative and performance overhead. Enforcement via the execution

environment provides the added benefit of being able to securely execute non-trusted legacy applications without duplicating security code already present in the operating systems.

PRIMA's enforcement mechanisms can control access to file and network connections. The majority of use cases in grid computing environments can be handled by controlling access to the file system (e.g. through file system access control lists and file system quota mechanisms). If combined with functions that can enforce access control on individual network connections, an even larger variety of access-control scenarios can be supported.

Execution environments can be provisioned in various ways. Among them are the following.

1. **Identity Authorization:** Basic identity credentials are mapped to preexisting execution environments configured with a static set of access rights. This enables the requested service to utilize all access rights associated with the execution environment.
2. **Mixed Mode Authorization**: Identity authorization is used to define an initial set of access rights. Additional access rights based on presented privileges are applied to the specific local execution environment before the request is served. After a resource access is completed the additional access rights are removed. Alternatively, the access rights could remain assigned to the local execution environment for the lifetime of the presented privileges.
3. **Privilege-based Authorization:** A request is served by a special-purpose, dynamically allocated and configured execution environment. The presented privileges are applied to a generic, initially very restricted, execution environment. After completion the execution environment is returned into its original, restricted state. Depending on the anticipated usage scenario the execution environment could be maintained and reused for future, similar service requests by the same subject.

The advantages of the PRIMA enforcement model are most beneficial if the chosen enforcement mechanisms fall in the privilege-based authorization category. However, there may be environments that will not allow for the dynamic allocation of execution environments, for example due to regulatory aspects, or power users may require statically created execution environments as they also perform interactive access to resources. The PRIMA model can accommodate these cases using mixed-mode authorization approaches and can provide improved functionality to environments with static allocation of execution environments.

3 Prima Architecture and System Components

3.1 Overview

The PRIMA system architecture and authorization process is illustrated in Figure 3-1. A dashed line separates the figure into two parts. The top part of the figure is the privilege

management layer which facilitates the delegation and selective use of privileges as discussed in Section 2. The bottom half of the figure is the authorization and enforcement layer represented by the PRIMA components discussed in this section. The actions in the authorization and enforcement layer occur at access time, that is, at or after the time that an execution request is made by a user.

The authorization and enforcement layer (bottom part of Figure 3-1) has two primary components. The first component is the PRIMA Authorization Module. The Authorization Module, described in detail in section 3.3, plays the role of a Policy Enforcement Point. The second component is the PRIMA Policy Decision Point which, based on policies made available to it, will respond to authorization requests from the PRIMA Authorization Module. The Policy Decision Point is discussed in more detail in section 3.4

Two other components in the authorization and enforcement layer are the Gatekeeper and the Privilege Revocator. The Gatekeeper is a standard Globus Toolkit [3] component for the management of access to Globus resources. It was augmented with a modular interface to communicate with the authorization components. This interface is described in section 3.2. The JobManager, also a standard component of the Globus Toolkit, has not been modified from the original Globus distribution. It is instantiated by the Globus Gatekeeper after successful authorization. It starts and monitors the execution of a user's job. The Privilege Revocator, explained in section 3.6, monitors the lifetime of privileges that were used to configure execution environments. On privilege expiration, the Privilege Revocator removes access rights and de-allocates the execution environment automatically. No manual intervention from system administrators is required.

The numbered arrows in Figure 3-1 indicate the steps in a typical access request and authorization sequence. Step 1, the delegation of privileges, happens prior to a request is issued. In Step 2, subjects select the subset of privilege attributes they hold for a specific (set of) grid request(s) and group these privileges with their short lived proxy credential using a proxy creation tool. The resulting proxy credential is then used with standard Globus job submission tools to issue grid service requests (Step 3).

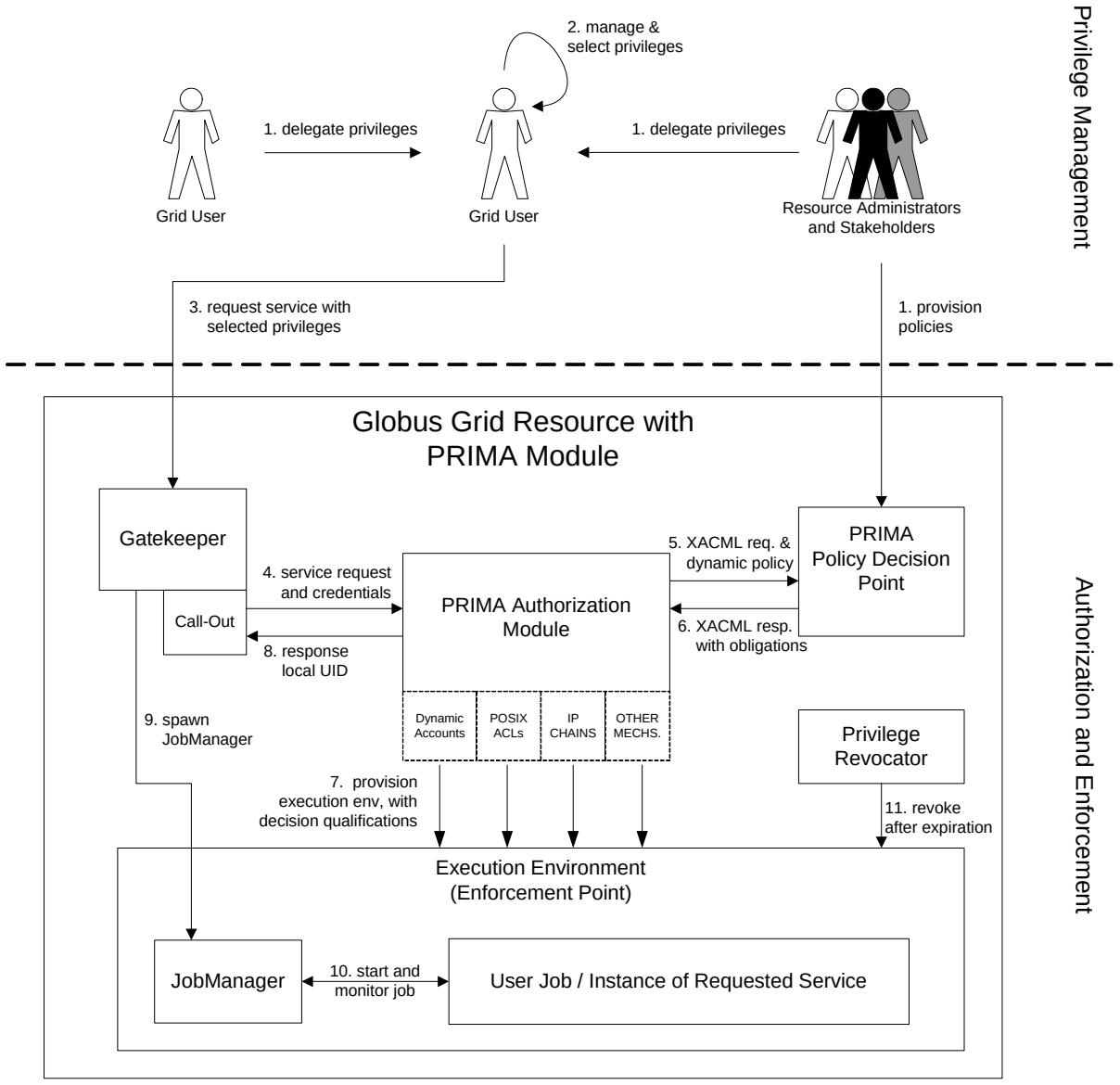

Figure 3-1 The PRIMA System Components and the PRIMA Authorization Process

Upon receiving a subject's service request, the gatekeeper calls the PRIMA authorization module (Step 4). The PRIMA authorization module extracts and verifies the privilege attributes presented to the Gatekeeper by the subject. It then assembles all valid privileges into a dynamic policy. To validate that the privileges were issued by an authoritative source, the Authorization Module queries the privilege management policy via the PRIMA Policy Decision Point (PDP). The multiple interactions between authorization module and PDP are depicted in a simplified form as a single message exchange (Step 5 and 6) in Figure 3-1. Once the privilege's issuer authority is established, the PRIMA Authorization Module formulates an XACML authorization request based on the user's service request and submits the request to the PDP. The PDP generates an authorization decision based on the static access control policy of this resource. The response will state a high-level permit or deny.

In the case of a permit response, the Authorization Module interacts with native and special purpose security mechanisms to allocate an execution environment (e.g., a UNIX user account with minimal access rights) and provision this environment with access rights based on the dynamic policy rules (Step 7). Once the execution environment is configured, the PRIMA Authorization Module returns the permit response together with a reference to the allocated execution environment (the user identifier) to the Gatekeeper and exits (Step 8). The following steps are unchanged from the standard Globus mechanisms. The Globus Gatekeeper spawns a JobManager process in the provided execution environment (Step 9). The JobManager instantiates and manages the requested service or process (Step 10).

In the case of a deny response, the Authorization Module returns an error code to the Gatekeeper together with an informative string indicating the reason for the denied authorization. The Gatekeeper in turn will protocol this error in its log, return an error code to the grid client (subject) and end the interaction.

The fine-grained privileges applied to the execution environment will remain active for their full lifetime as specified in the original privilege. The Privilege Revocator watches over the validity period of dynamically allocated user accounts and all fine-grained access rights, revoking them when the associated privileges expire (Step 11).

3.2 The Globus GRAM Authorization Call-Out

The Globus Gatekeeper has been extended with two interfaces, one of which – the identity mapping interface – is used in PRIMA. This interface has been designed in collaboration with members of the Particle Physics Data Grid, the European Data Grid, and the Globus Alliance. Within the PRIMA research this interface has been implemented for the Globus Toolkit Version 2.2.4 and has been deployed in other Grid projects [14]. The Globus project has later produced its own implementation of the interface (with slight changes in the way parameters are exchanged). The most recent release of the Globus Toolkit at the time of this writing (GT3.2) features this call-out as a standard component.

The identity mapping interface replaces the static grid-map file mechanism by which the grid user identity (the subjects distinguished name extracted from the X.509 identity certificate used by the client during authentication) is mapped to a local user identity at the resource. Using the identity mapping interface, sites can replace the default behavior controls. The gatekeeper will call the identity mapping interface only one time (multiple identity mapping modules are not supported).

The PRIMA implementation adds code for parsing a call-out configuration file, loading and initializing modules, and logging to the existing Globus gss_assist library. A configuration file allows a maximum of one identity mapping module specified by a “gridmap” entry (if none is specified the implementation will revert back to the standard grid-map mechanism) and zero or more admission control modules can be listed using the “authz” entry type. A sample configuration file is shown in Figure 3-2, it identifies a dynamically loadable library “prima”

```
gridmap          prima
authz            vtauthz1
authz            vtauthz2
```

Figure 3-2 Call-out ConfigurationFigure 3-2 Call-out

for use by the identity mapping call-out and two dynamically loadable libraries “vtauthz1” and “vtauthz2” to be called by the admission control interface.

The Globus gatekeeper’s program flow is slightly different with the call-out in place. The traditional gatekeeper performed the mapping to a local user identity before accepting the service request from the client. The modified gatekeeper must read the service request and extract the service name before the authorization modules are called in order to provide the service name with the parameters as specified in the interface.

3.3 PRIMA Authorization Module

As shown in Figure 3-1, the PRIMA Authorization Module interfaces the Globus gatekeeper with the authorization decision and enforcement mechanisms of the PRIMA system. It is implemented as a dynamically loadable library (libprima.so) and called from the Globus gatekeeper after a service request has been received from an authenticated client. The authorization module performs the following sequence of actions:

- validates the privileges accompanying the request and requests an authorization decision,
- determines the local user account that will be used to service the request, and
- provisions the selected user account with the access rights need to perform the requested service based on the validated privileges.

These actions are explained in the following subsections.

3.3.1 Validating Privileges and Requesting an Authorization Decision

The PRIMA authorization module receives from the Globus gatekeeper a Generic Security Services (GSS) context containing all the certificates used during mutual authentication including the privilege attributes that were provided by the requesting client. The GSS context is a container for security related information whose contents can be accessed through a defined programming interface (GSS-API) [15]. This interface abstracts away the low-level authentication token formats and cryptographic mechanisms by which the context is implemented. The PRIMA implementation further extends the GSS-API to abstract authorization token formats (i.e. the Attribute Certificate format used in PRIMA) and enables the retrieval and verification of privilege attributes in a mechanism independent format. The Globus GSS-API implementation is based on the security functions provided by OpenSSL’s [16] implementation of the Transport Layer Security (TLS) protocol and the OpenSSL cryptographic library (libcrypto).

The authorization module extracts and verifies the privileges contained in the GSS context. The GSS-API extensions developed as part of the PRIMA research provide for the verification of the privilege attributes by extracting the attribute certificates and issuer identity certificates that are embedded in the subject’s proxy certificate and constructing and verifying the certificate chain for each one of those attribute certificates. Figure 3-3 provides an overview of the certificate chain verification process for the proxy certificate used for authentication (top part, performed by the standard Globus implementation of the GSS-API) and for every attribute certificate that is provided (bottom part, provided by the PRIMA

extensions to the Globus GSS-API implementation). The figure assumes a single proxy certificate is used (only one direct delegation from the identity certificate to the proxy). Multiple delegation steps (another proxy derived from this first proxy) would add additional proxy certificates. If a hierarchy of Certification Authorities is used then multiple CA certificates are also possible.

The basic validation process proceeds as follows. This process is the same for both proxy and attribute certificates. Each certificate's signature has to be verified using the issuer's certificate public key. The issuer field or a certificate must match the subject field of the upstream (issuer's) certificate. All certificates in the chain must be within their denoted validity time at the time of validation, i.e. the certificate with the earliest closest expiration date denotes the maximum lifetime for all certificates that are downstream from it. Finally the holder of an attribute certificate must be the end-entity (a.k.a. the user or subject) that authenticated by means of the proxy certificate and associated private key (the proxy certificate issuer).

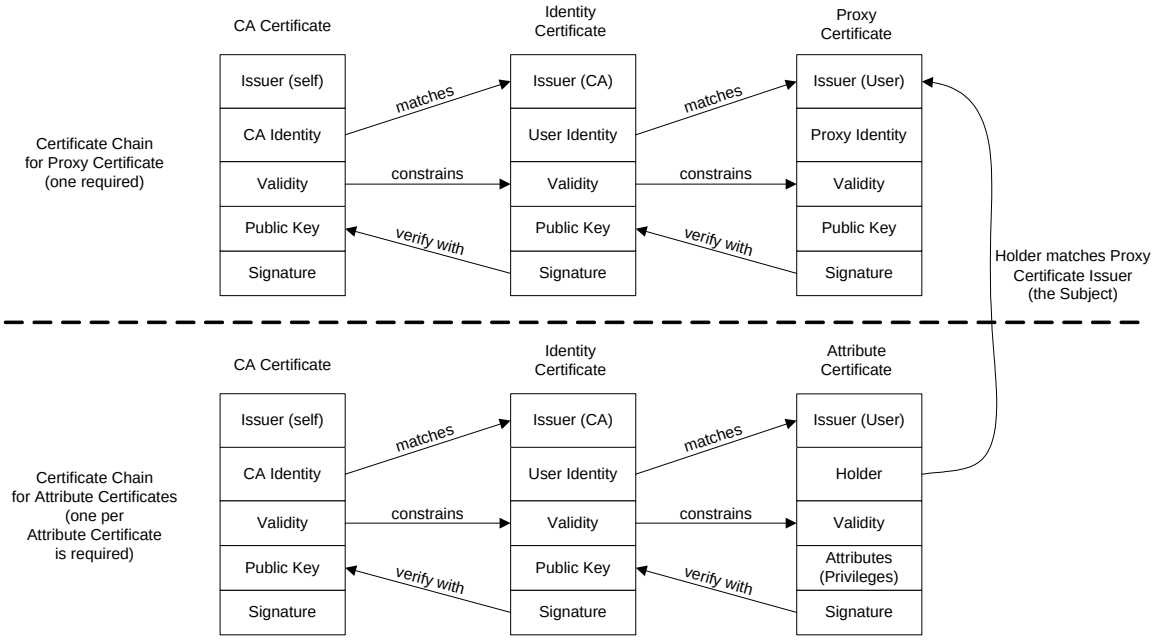

Figure 3-3 Proxy and Attribute Certificate Validation

Once all privilege attributes have been validated the PRIMA authorization module determines if the issuers of the privilege attributes were authoritative to issue the specific privileges for the resources named in the privileges. To perform this check, the authorization module generates XACML queries that are transmitted to the PRIMA Policy Decision Point. The PDP evaluates the queries against the applicable privilege management policies. An example or such a request is shown in Figure 3-4. The request consists of a subject tag that names the privilege issuer, a resource tag that specifies to what resource the issued privilege applies, and an action tag that indicated that the issuance of a privilege of a specific type must be authorized. Figure 3-5 shows the corresponding (positive) response which consists of a result tag with a decision element and an optional status code tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request    xmlns="urn:oasis:names:tc:xacml:1.0:context"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context
        cs-xacml-schema-context-01.xsd">
    <Subject>
        <Attribute
             AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
              DataType="http://www.w3.org/2001/XMLSchema#string">
            <AttributeValue>/CN=Markus Lorch/OU=Computer Science/O=Virginia Tech
/ST=Virginia/C=U</AttributeValue>
        </Attribute>
    </Subject>
    <Resource>
        <Attribute
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
              DataType="http://www.w3.org/2001/XMLSchema#anyURI">
            <AttributeValue>http://zuni.cs.vt.edu/opt/prima-vo/analysis-results-
2004</AttributeValue>
        </Attribute>
    </Resource>
    <Action>
         <Attribute
               AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
               DataType="http://www.w3.org/2001/XMLSchema#string">
            <AttributeValue>IssueFilePrivilege</AttributeValue>
         </Attribute>
    </Action>
</Request>
```

Figure 3-4 XACML Request to Verify Issuer Authority

```
 <Response     xmlns='urn:oasis:names:tc:xacml:1.0:context'
   xsi:schemaLocation='urn:oasis:names:tc:xacml:1.0:context
     http://www.oasis-open.org/tc/xacml/1.0/sc-xacml-schema-context-01.xsd'>
   <Result>
     <Decision>Permit</Decision>
   <Status>
     <StatusCode Value='urn:oasis:names:tc:xacml:1.0:status:ok'></StatusCode>
   </Status>
   </Result>
 </Response>
```

Figure 3-5 XACML Response

For some fine grained privileges the privilege management policy will not contain rules that govern who may issue such privileges. An example of this situation is file access privileges. To be authoritative to issue a file access privilege the issuer must be the owner of the file. Basing authority on file system ownership has two implications for the verification of file access privilege:

1. The PDP cannot determine the authority of the issuer and the authorization module does not understand the detailed semantics of the file system ownership mechanisms (they are different based on which file access enforcement mechanism is being used). In these cases the verification of the authority of the issuer is deferred until the enforcement mechanism attempts to implement the access right contained in the privilege.
2. The enforcement mechanism may limit the set of files for which issued privileges can be verified. This limitation is due to the need for a reverse mapping of the user account owning the file to the external grid identity issuing a privilege. If a file is

owned by a dynamic user account then this ownership is lost when the dynamic user account is deallocated. Thus, a mapping to a grid identity is not possible. Some enforcement mechanisms (e.g. SlashGrid, see 3.5.2) do not rely on this mapping to identify the owner and can support the creation and verification of privileges for file objects owned by dynamic user accounts.

The use of ownership information provided by the enforcement mechanism is advantageous because it enables privilege management policies to be smaller, and focused on coarser grained issues. The privilege management polices need only be concerned, for example, with who may delegate a system access privilege or to whom privileges in general may be delegated (e.g. only to other members of the same organization). Furthermore, it prevents the otherwise redundant storage of security information and improves the integrity of the system. The distributed storage of authority information does not prevent the user from delegating rights to files for which the user is authoritative. It simply means that the privilege management policy by itself cannot validate such privileges.

If the enforcement mechanisms detect that a privilege cannot be applied to an execution environment the affected privileges will be silently ignored and the execution environment configured with only those access rights based on privileges from authoritative entities. This opportunistic approach was chosen because the ignored privileges may not or no longer be required to successfully complete the request (e.g. due to changes in the service code). It would be a trivial change to the semantics of the authorization module to abort the process of provisioning access rights to an execution environment. This approach would be motivated in systems where the cost of failed service executions due to missing privileges is believed to be higher than the benefits of this opportunistic approach. Checkpointing and recovery mechanisms can also be utilized to mitigate the effect of missing privileges (see notes to mitigate late-binding effects below). The decision on which of these two approaches to follow could also be decided by the system administrator and encoded as part of the resource's policy.

The combination of all valid privileges constitutes a dynamic policy which will be used to construct and configure the execution environment for the requested service. Once the authoritativeness of all privilege issuers is validated, a final authorization request to the PRIMA PDP is made. This authorization request ensures that the service request is allowed by the access control policies configured into the PDP. This final request enables resource administrators to overrule access rights assigned via privileges and ensures that administrators have final control over resource usage.

The PRIMA model calls for the dynamic policy to be provided to the PDP and the resource request to be evaluated against the combination of the dynamic policy and static access control policies. The PRIMA system implementation, specifically the Authorization Module, directly makes a decision on the applicability of the provided privileges to the resource request. As privileges contain access rights that can be exercised without further interpretation the authorization module can provision these access rights directly to the enforcement functions (explained in detail in sub-section 3.3.3). The enforcement functions will then make individual access control decisions when access to the resource object is requested by the process that serves the subject's request. If a privilege was presented for this type of access to the resource object the enforcement mechanisms will grant access. If no privilege was presented access will be denied. This late-binding approach of decision making

may prevent some services from completing if a required privilege was not provided prior to execution of the requested service. At the same time, late-binding provides a solution to the level of detail mismatch problem discussed in section 2.3. This problem stems from the inability of a decision function to understand what specific resource requests an executable will make once it has been allowed to start. To mitigate possible impacts on long running applications that cannot gracefully accommodate denied access to operating system resources checkpointing and recovery solutions such as those developed in the Weaves project [11] may be utilized to suspend executables that discover a lack of access rights until the appropriate privileges can be acquired and provided to the resource by the subject.

The fact that access control decisions on fine grained access rights are deferred until runtime does not affect the ability of the resource access control policy to constrain rights granted via privileges. If the access control policies do not allow for the access in general or have detailed instructions for the enforcement mechanisms on how to constrain execution of the request, this information can be included in the final authorization decision response (see final authorization decision request above) provided to the authorization module by the PDP.

3.3.2 Mapping to a Local User Account

For each authorized request, PRIMA determines a local user account under which the requested service will execute. The user account may already be allocated to the subject (traditional static user accounts) or may be dynamically allocated by PRIMA. Dynamically allocated accounts are drawn from a pool of accounts made available to PRIMA by the system administrator. These accounts disallow direct login and have minimal initial rights. Such accounts may also have lifetime restrictions allowing them to serve only the current request or to exist only for a specified period of time.

The logic of the creation, configuration, and management of user accounts by the PRIMA authorization module is outlined in Figure 3-6. System access privileges are used in PRIMA to convey a right to use a dynamically allocated account. After verifying the validity of a system access privilege, the authorization module check determines the privilege's applicability by verifying that the host name specified in the privilege matches the name of the resource. If the privilege is applicable to the resource, PRIMA will check for an existing dynamically allocated user account matching the distinguished name and an optional group or project identifier that are contained in the privilege. This check is used to distinguish among multiple concurrent dynamic accounts held by the same entity. An existing dynamic account is chosen if the check succeeds. At this time, an extension can be made to the lifetime of the account if the privilege has a longer lifetime than the chosen account. If no existing mapping to a dynamic account is found, a new account is assigned from the pool of available accounts. The system also supports the use of a static account if no system access privilege was presented with the request. In this case the traditional grid-map file is consulted to see if a static local user account exists for the user identity established during authentication.

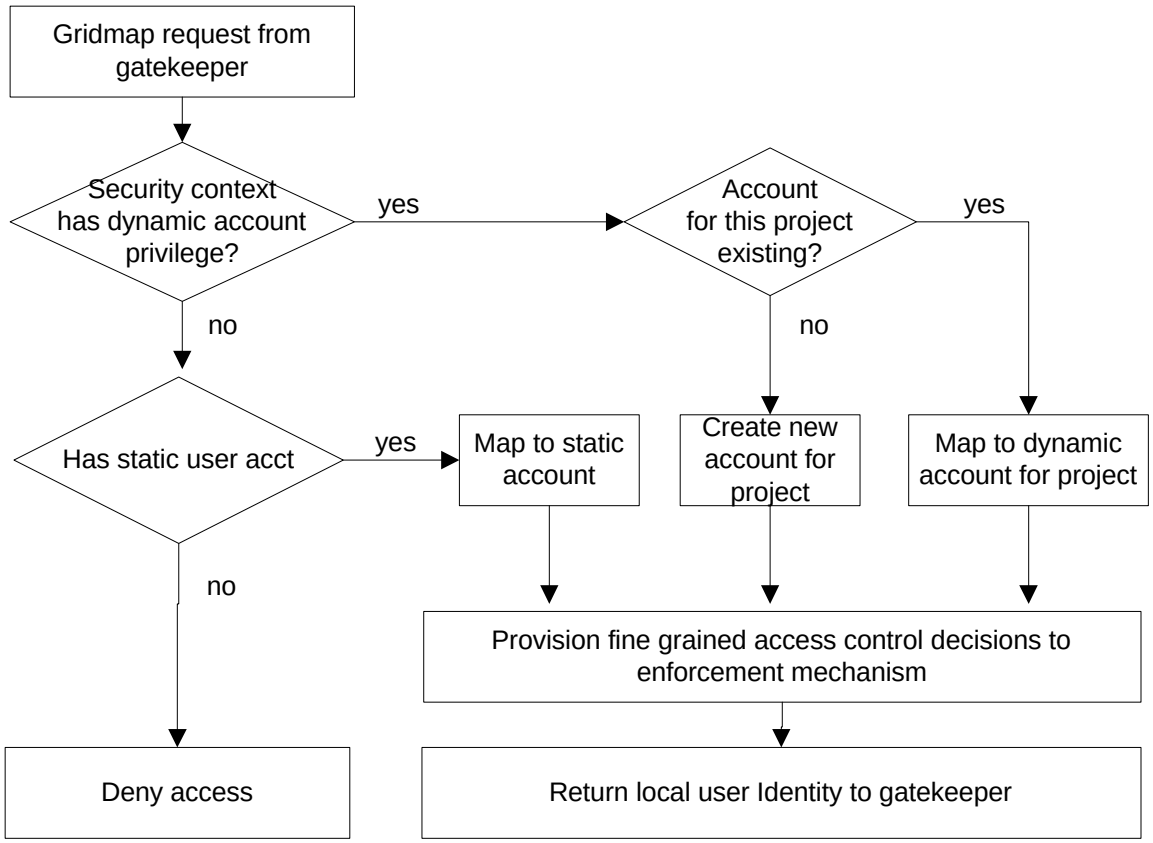

Figure 3-6 Account Mapping and Allocation Logic

Management of dynamic accounts is controlled through a privilege database storing the user's identity, project assignment, and lease expiration for each account. Accounts remain valid until the lease expiration date is reached. When an account's lease expires it is automatically cleansed (all access rights associated with the local user identity are reset) and returned to the dynamic user account pool. The administrator retains control over the number of accounts available at any given time and can manually change or revoke account assignments.

3.3.3 Provisioning Privileges to the Enforcement Mechanisms

Before the determined local user identity is returned to the Globus gatekeeper (see Figure 3-1, Step 8), the fine-grained access rights contained in the dynamic policy are provisioned to the enforcement mechanisms (see Figure 3-1, Step 7). For each fine-grained access right the enforcement mechanism is also provided with the issuer of the corresponding privilege attribute. This enables the enforcement mechanisms to make discretionary authority decisions based on the owner of an object. As noted above, the privilege management policy will not include authorization statements for every single file on a file system. Instead, the authority to delegate a file access privilege is defined by ownership of the file (which is only known to the file system mechanism). A file owner is authoritative for his files unless the privilege management policy overrules this mechanism with an explicit rule that defines a different behavior.

The PRIMA authorization module can interface with a variety of native or special purpose enforcement mechanisms on the local resource. This property allows the PRIMA mechanisms to be implemented on heterogeneous platforms. The enforcement mechanisms currently used are explained in section 3.5.

3.4 The PRIMA Policy Decision Point

The PRIMA Policy Decision Point (PDP) is a policy engine for XACML policies. It is shown to the right of the PRIMA authorization module in Figure 3-1. The PDP responds to authorization requests made by the authorization module. The PDP accepts a set of XACML policies and an XACML authorization request as input. It will evaluate the request with respect to the presented policies and produce an XACML response.

The PRIMA PDP is currently implemented as a wrapper around a freely-available (in binary) implementation by JiffySoftware [17] called the Jiffy Policy Tester. Communication between the PRIMA authorization module and the PRIMA PDP is achieved via temporary text files that are created dynamically and contain the XACML requests and responses. While the Jiffy Policy Tester was designed as a command line tool for policy schema evaluation (hence the need to communicate with it thought text files) it provides sufficient functionality to sequentially evaluate XACML requests against policies for a single PEP.

For added flexibility and manageability a site-centralized decision point could be used though drawbacks in reliability and performance may be incurred. A centralized decision point can contain policies that apply to a set of resources without the need to replicate such policies on every resource. Drawbacks of this centralization include the introduction of a single point of failure through the centralized component, and the need for secure message exchange between the PRIMA authorization module and the PDP (e.g. via the SAML protocol [18]). Furthermore, the Jiffy Policy Tester would not be able to handle the load of multiple resources requesting authorization decisions simultaneously, nor would the current solution of passing requests and responses via text files be adequate. A more advanced policy decision point could be built using Sun Microsystem's open source XACML library [19]. Sun’s implementation provides richer functionality, is easily available as open source software, and allows custom communication mechanisms for more flexible communication with a PDP process, for example through network connections.

3.5 Enforcement Mechanisms

PRIMA uses native enforcement mechanisms to control access to files and network connections. PRIMA currently utilizes two enforcement mechanisms to control file access: POSIX.1E file system access control lists [20] which are available on Unix operating systems, and XML-based Grid Access Control Lists (GACLs) which were developed as part of the SlashGrid grid-aware filesystem framework [21]. The use of POSIX access control lists is described in more detail in section 3.5.1 and the SlashGrid mechanism in section 3.5.2. An enforcement mechanism for controlling access to network connections that has been explored for the PRIMA system is iptables [22]. Section 3.5.3 elaborates on the network enforcement mechanism.

3.5.1 File System Access Control Lists

Secure sharing of files is often more limited than what is needed by many grid usage scenarios. Traditional operating systems require the creation of user group accounts and the changing of group membership for users as well as files. Standard UNIX-like systems store file access permissions (read (r), write (w), execute (x)) for three categories of users: the file owner, the file group, and others. The management of access rights by assigning them to one of the user-group-other categories poses a problem if multiple users, that are not member of file group, require access. A file cannot be assigned to several groups.

```
# file: data/simresult
# owner: mlorch
# group: csstudent
user::rw-
user:abazaz:r-
user:kafura:r-
user:akarnik:rw-
group::r--
mask::rw-
other::—
```

Figure 3-7 A File System ACL

File system access control lists remove these limitations by extending the standard file permission categories to allow for finer control. In addition to the file owner, file group, and others, additional users and groups can be granted or denied access. Figure 3-7 shows a sample text representation of an ACL specifying additional access permissions for users abazaz, kafura and akarnik for a file owned by user mlorch. The basic UNIX access permissions in this example enable the owner (mlorch) to read and write to the data file “data/simresult”. The additional users “abazaz” and “kafura”, as well as every user that is a member of the group “csstudent” can read the data while the user “akarnik” is given read and write permission. The mask entry defines the maximum rights that can be given to additional users (beyond user, group, other) by the ACL. Other users of the system (members of the “other” category) have no access to the file.

In PRIMA, the system’s ACLs are modified dynamically based on file privileges provided by the authorization module to configure the set of access rights for an execution environment. The ACL is changed only when the privilege issuer is the file owner or an entity explicitly named as authoritative by the privilege management policy (see related discussion on delayed issuer verification in section 3.3.1). When an ACL entry is dynamically modified the change is recorded in a resource privilege database together with the issuer and expiration information. This database is used for subsequent restoration of the original state of the ACLs (see Privilege Revocator explanation in section 3.6).

The authorization module uses system calls to read and modify ACLs. POSIX ACLs (on Linux) are stored as extended attributes directly associated with the inode of the file or directory to which they correspond. Extended attributes are atomic objects and can only be read and replaced as a whole. Modifications to extended attributes (and thus file system ACLs) have to be done in a buffer by the application and the new version written back to the file system in a single write access.

ACLs are available on a number of UNIX operating systems, specifically Linux, Solaris, and IRIX. On these systems, ACLs are implemented following loosely the POSIX.1E [20] draft recommendation. The specific file systems that support ACLs on Linux to date are EXT2, EXT3, NFS, ReiserFS, IBM JFS, SGI XFS. Unfortunately, the POSIX.1E

recommendations have never been formally completed and standardized. Interfaces and implementations in prevalent operating systems differ slightly but exhibit the same core functionality. The core system calls differ only slightly between Linux and Irix systems and would allow this PRIMA enforcement mechanism to be easily ported to Irix. Solaris, which implements ACLs based on an earlier draft of POSIX 1.e differs significantly. To achieve a truly portable implementation an abstraction layer could be used to hide the differences in implementations.

3.5.2 Grid Access Control Lists

Grid Access Control Lists (GACLs) are an XML representation of file access control lists with additional features to accommodate users belonging to different groups or projects. Figure 3-8 shows a simple GACL specifying the issuer, user, and associated permissions. The SlashGrid framework [21] uses GACLs in a grid-aware file system. Subject privileges are associated directly with the subject's distinguished name rather than a local user identity and thus can provide for consistent, long-term privilege assignments even if local user identities change over time.

The PRIMA enforcement mechanism based on GACL and SlashGrid is similar to the mechanism that uses POSIX ACLs. It verifies that the issuer is authoritative for the file or directory tree to which a privilege applies to (i.e. the issuer has the "admin" right on the object) and then applies the access right by modifying the corresponding GACL. All modifications are also stored in a resource privilege database (for later restoration).

```
<gacl version="0.0.1">
<entry>
  <person>
   <dn>/CN=Markus Lorch/O=vt/C=US</dn>
  </person>
  <allow>
    <write/><admin/><read/><list/>
  </allow>
</entry>
<entry>
  <person>
   <dn>/CN=Sumit Shah/O=vt/C=US</dn>
  </person>
  <allow>
    <read/><list/>
  </allow>
</entry>
</gacl>
```

Figure 3-8 A Grid Access Control List

One specific advantage of the GACL format is that it stores access rights associated directly the grid identity (the distinguished name that was established during authentication based on the proxy certificate chain). This simplifies the verification of a privilege issuer as no reverse mapping from a local user account (that holds file ownership) to a grid identity has to be performed to verify the authority of a file privilege issuer.

SlashGrid provides a server process that enforces the rules expressed in GACL files. SlashGrid uses the CODA [23] kernel module in Linux to intercept access to files in an emulated file system, consult the GACL rules, and, if appropriate, forward the access request to the underlying file system. GACLs are stored in a separate text file in the directory for which they specify access control rules (".gacl" file). If access control rules are specified for an individual file, the corresponding GACL is stored in the same location as the governed file and the naming convention is "filename.gacl". The GACL library provides functions to create and parse GACL files.

3.5.3 Host-Based Firewall Rules

To support the selective control over network access of services running in an execution environment the PRIMA system could be extended to use the iptables packet filtering framework [22] of the Linux 2.4 and 2.6 kernels. Iptables enables the specification of network filtering rules based on the local user identity associated with the network packets. This additional enforcement mechanism enables PRIMA to selectively allow or disallow the creation of network connections. A dynamically allocated execution environment could not send network packets to external addresses unless an iptables permit rule for the execution environment's user account is present.

These network firewall rules are enforced directly by the operating system kernel. The enforcement of network rules for incoming connections (i.e. a requested service listening to a network port) is not supported in this manner with the current version of iptables.

The network rules are dynamically configured by modifying the iptables internal state. Such modifications are made by calling the iptables command with appropriate parameters. The modified state is also written to the iptables boot script to retain the changed rules in case of a system reboot. The modified state is also logged in the privilege database together with its expiration time.

3.6 The Privilege Revocator

The Privilege Revocator, present on each PRIMA enabled resource, automatically revokes privileges and dynamic user accounts when they expire. The resource's privilege database is periodically checked by the Privilege Revocator. When an individual fine-grained privilege expires, the Privilege Revocator removes this right from the respective access control list. In the case of privileges for a dynamic user account, a notification can be sent to the account holder before expiration to allow the account holder time to present a new privilege for the account to extend the account lease and keep the data and other privileges associated with this account. Expired accounts are cleansed of all access rights, returned to its initial state, and made available for reallocation.

4 Comparison and Conclusions

PRIMA is related to a number of other systems including PERMIS [24], AKENTI [25], CAS [26], and CARDEA [27]. These systems uniformly focus on the improvement of the management of access rights for large, relatively stable and longer lived user communities. PERMIS implements a role based access control (RBAC) scheme in which rights are associated with roles rather than with specific entities. In an RBAC scheme, access is granted by rendering role and other descriptive attributes against applicable access control policies (role attributes thus grant access rights indirectly). The advantages of RBAC lie in the increased manageability of large user bases, especially if many users share a common and stable set of access rights. Akenti [25] is an access control system that provides distributed policy management mechanisms. Multiple stakeholders can individually and independently of other stakeholders define resource use conditions that must be satisfied by a requestor before access is granted. The Akenti system will combine these conditions from all authoritative stakeholders when making authorization decisions. The Community

Authorization Server (CAS) [26] reduces administrative overhead imposed by virtual organizations by separating the administration of resource policies from community policies. Resource administrators grant bulk rights to a grid community (e.g. a VO) and community administrators then decide what subset of a community's rights an individual member will have. Group members authenticate to grid resources with a group credential that has restrictions applied to it (limited proxy credential) limiting the individuals rights to a subset of the rights the community has at the resource. The limitations imposed on the rights the group account has at the resources are enforced by the grid service application. Cardea [27] is an authorization system that focuses on separating authorization logic from local identities at the resources. Cardea policies are defined with respect to high level identities such as entity's distinguished names and thus enable the management of rights uniformly across a variety of heterogeneous systems with varying local identity management mechanisms. Authorization decisions depend heavily on the attributes a service requestor holds (e.g. group membership and clearance level).

PRIMA improves on existing security mechanisms through its use of secure privileges to enable least-privilege access, through dynamic policies that improve scalability while preserving the ultimate control of resource administrators, and through dynamic execution environments that yield more secure enforcement of fine-grain authorization decisions.

Through secure privileges PRIMA provides flexible rights management that enables least privilege access and that can stand alone or be combined with other systems. PRIMA achieves flexible, grid-layer management of access rights through the externalization (abstraction) of system-specific access rights. The advantages of a privilege-based system lie in the considerable flexibility provided to the user who can be given the authority to delegate privileges to others (thus enabling many collaborative scenarios) and to use only the least privileges necessary for a specific access (increasing security through least privilege access). PRIMA uses existing security languages (XACML) and security credential standards (X.509 attribute certificates) and thus enables the integration with other security services such as PERMIS and CARDEA. Both PRIMA and PERMIS leverage X.509 attribute certificates, while CARDEA and PRIMA both use XACML to define policies. Current work on PERMIS [personal communication with D.W. Chadwick] includes the replacement of the proprietary policy language used in PERMIS with XACML. The use of standards enables the combination of authorization systems, for example attribute repositories such as VOMS [28] (see discussion on current integration with VOMS below) can provide the advantages of role-based management of access control for large communities, while PRIMA privileges enable the formation and management of small and transient communities.

Through dynamic policies PRIMA improves the scalability of security services without reducing the ultimate control of resource administrators. The scalability is achieved by allowing users to delegate privileges to each other and to assign selected privileges they hold to specific tasks, processes or services. Other authorization systems like those discussed above typically require the deployment of community specific infrastructure components (such as a CAS Server). Such infrastructure deployments are not required by PRIMA. Privilege delegation is easily enabled, disabled or constrained in PRIMA by resource administrators via a privilege management policy. This policy governs who is authoritative for specific resource objects and defines within what bounds the delegation of privileges must occur. This policy gives the resource administrator fine-grain control over the

distributed management of privileges. The relationship between externally managed privileges presented by a user when requesting a service and the authorization policy defined for the resource being accessed is captured in PRIMA's concept of a "dynamic security policy". This policy is "dynamic" since it applies only for the evaluation of the user's current request. The dynamic policy insures that the user's manipulation of rights is consistent with that granted by resource administrators.

Through dynamic execution environments PRIMA improves the enforcement of fine-grain rights and supports legacy applications. PERMIS, Akenti, CAS, and Cardea enforce access control decisions at the application level. This approach depends on custom application code that is trusted to restrict its resource usage in accordance to the authorization decisions made by the policy decision function. In contrast, PRIMA relies on the dynamic creation, configuration and management of an execution environment configured with the minimal set of fine-grained access rights required for the specific access. Enforcement via this execution environment provides for the secure execution of non-trusted legacy applications without the need to duplicate security code already present in the operating systems. Experimental work on similar dynamic runtime environments for grid services that are developed for the Open Grid Services Architecture (OGSA) [29] is documented in [30] and may be combined with the PRIMA enforcement mechanisms for deployment in OGSA environments.

The possible use of PRIMA's enforcement functions in combination with other authorization systems furthermore demonstrates the flexibility of the PRIMA components. The architectures of the systems discussed all follow the separation of the authorization tasks into those associated with making an authorization decision and those associated with enforcing such a decision (see [31][32]). This enables the use of one system's enforcement functions (e.g. PRIMA execution environments) with the decision functions of another (e.g. role-role based policy decision making). A standard interface between the policy enforcement policy decision points is being developed at the Global Grid Forum. This interface is partially incorporated in the development versions of PERMIS, the Globus Toolkit 3.3 and PRIMA. To embed advanced authorization mechanisms into existing grid middleware an authorization call-out interface is specified. Together with members of a number of large grid computing projects, an interface specification was produced and subsequently implemented as one component of PRIMA. The interface implementation has been deployed in other grid projects, and included in the Virtual Data Toolkit (VDT), a base distribution for many grid projects.

PRIMA's mechanisms can leverage community infrastructure to provide improved support for large, long-lived communities. Current work is integrating PRIMA decision and enforcement functions with the Virtual Organization Management Service (VOMS) [28], a community centric authorization attribute server that aids in the management of larger virtual organizations by providing information about group membership. VOMS issues subject attributes (as X.509 attribute certificates) to members of the community. Subjects then select which group membership attribute they want to supply with a grid service request the same way they group PRIMA privileges with a request. The PRIMA authorization module is used to extract this group membership information from the security context. The PRIMA PDP then takes these attributes into consideration when making policy decisions. This adds the ability to manage larger groups based on VOMS attributes while smaller groups can be managed using privileges. This work is part of a national scale grid computing project (U.S.

CMS) involving multiple organizations under the lead of Fermi National Accelerator Laboratory.

This research lays the foundation for significant future work. This work includes combining privilege-based and role-based models as mentioned above, determining least-privilege access requirements from service descriptions, assessing the costs and risks when privileges cannot be perfectly mapped to the enforcement layer, and devising means to securely convey resource policies in a distributed environment.

Determining the least privileges needed to use a service may be achieved by analyzing service descriptions. Service descriptions are represented in special-purpose languages (e.g., the web service description languages, WSDL) or in application-specific meta-data. An issue is how to use the security information inferred from such descriptions to create the minimal access rights necessary to use the service. A solution to this problem is needed to help non-technical users manage their assets through the delegation, restriction and review of access to resources under their control.

Risk-based and cost-based factors need to be studied for those cases where fine-grain privileges cannot be perfectly mapped to the enforcing system's access control mechanisms. In the PRIMA model, fine-grained access-control decisions are based on explicit resource policies and user privileges. These decisions are enforced at heterogeneous resources by mapping the privileges to access rights on the resources using the resources' security primitives. There may be cases where a one-to-one mapping is not possible due to differences between the granularity of the privilege and the granularity of the primitives at the resource. For example, a privilege allows append access to a specific file, but the resource can only support simple read or write access control. In such cases, it may be desirable to make a risk-based and cost-based assessment of whether to authorize the access. A risk assessment and analysis of the enforcement overhead associated with finer grained enforcement than what is possible using the existing security primitives at the resource would provide the basis for such an assessment.

This paper presents the PRIMA system for privilege management, authorization and enforcement in grid environments. This work is particularly motivated by the desire to support spontaneous, short-lived collaborations among small groups of grid users. Beyond this immediate purpose, PRIMA's improved security services contribute to the development of scalable grid environments and are also applicable in other distributed application domains. The feasibility of the PRIMA approach is demonstrated by an implementation of the PRIMA model that extends the Globus Toolkit. The PRIMA enhanced Globus system supports a variety of collaborative scenarios not possible in the standard Globus system.

5 References

1. I. Foster, C. Kesselman, and S. Tuecke, ”The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, International Journal of Supercomputer Applications, 2001.
2. I. Foster, C. Kesselman; G. Tsudik, and S. Tuecke “A Security Architecture for Computational Grids”, Fifth ACM Conference on Computers and Communications Security, November 1998
3. I. Foster, and C. Kesselman, “Globus: A Toolkit-Based Grid Architecture” in “The Grid, Blueprint for a Future Computing Infrastructure”, I. Foster, and C. Kesselman, Editors, Morgan Kaufmann, San Francisco, 1999, pp. 259-278
4. Cal Ribbens, Dennis Kafura, Amit Karnik, Markus Lorch, "The Virginia Tech Computational Grid: A Research Agenda", Virginia Tech Technical Report TR-02-31, December 2002, http://eprints.cs.vt.edu:8000/archive/00000641/
5. M. Lorch, D. Kafura, “Symphony – A Java-Based Composition and Manipulation Framework for Computational Grids”, In Proc. Second Int. Symposium on Cluster Computing and the Grid, Berlin, Germany, May 2002
6. D. Agarwal , B. Corrie, J. Leigh, M. Lorch, J. Myers, R. Olson, M. E. Papka, M. Thompson “Security Requirements of Advanced Collaborative Environments”, Global Grid Forum Informational Document Draft
7. S. Mullen, M. Crawford, M. Lorch, D. Skow, “Site Authentication, Authorization, and Accounting Requirements”, Global Grid Forum Informational Document Draft
8. Markus Lorch, Dennis Kafura, "Grid Community Characteristics and their Relation to Grid Security", Technical Report TR-03-20, Computer Science, Virginia Tech, June 2003, http://eprints.cs.vt.edu:8000/archive/00000678/
9. Simon Godik, Tim Moses, et al, “eXtensible Access Control Markup Language (XACML) Version 1.0”, OASIS Standard, February 18th, 2003
10. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, “A Fine-Grained Access Control System for XML Documents”, in ACM Transactions on Information and System Security (TISSEC), vol. 5, n. 2, May 2002, pp. 169-202.
11. S. Varadarajan, N. Ramakrishnan, Novel Runtime Systems Support for Adaptive Compositional Modeling in PSEs, Future Generation Computing Systems (Special Issue on "Complex PSEs for Grid Computing"), 2004, to appear, http://people.cs.vt.edu/~ramakris/papers/pseruntimesupport.pdf
12. I. Goldberg, D. Wagner, R. Thomans, and E. Brewer “A secure environment for untrusted helper applications”, Proceedings of the Sixth USENIX UNIX Security Symposium, July 1996
13. Virtual Executing Environment, http://www.intes.odessa.ua/vxe, visited 2004-04-04
14. V. Sehkri, I. Mandrichenko, D. Skow, “Site Authorization Service (SAZ)”, Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, CA, USA, March 2003, available from http://arxiv.org/pdf/cs.DC/0306100
15. J. Linn, "The Generic Security Service Application Program Interface, Version 2", Internet RFC2078, Internet Engineering Task Force, Network Working Group, January 1997
16. http://www.openssl.org, visited 2004-04-04
17. http://www.jiffysoftware.com, visited 2004-04-04
18. P. Hallam-Baker, E. Maler, et al, “Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML), Oasis Standard, November 5th, 2002
19. http://sunxacml.sourceforge.net, visited 2004-04-04
20. IEEE standard portable operating system interface for computer environments”**,** Withdrawn IEEE Draft Standard 17, Posix 1003.1, 1988, http://wt.xpilot.org/publications/posix.1e

21. A. McNab, “SlashGrid – a Framework for Grid Aware Filesystems”, http://www.gridpp.ac.uk/authz/slashgrid/, visited 2004-03-03
22. http://www.netfilter.org, visited 2004-04-04
23. M. Satyanarayanan “Mobile Information Access”, IEEE Personal Communications, Februar 1996, pp. 26-33
24. D. W. Chadwick, O. Otenko "The PERMIS X.509 Role Based Privilege Management Infrastructure" in proc. of the 7th ACM SYMPOSIUM ON ACCESS CONTROL MODELS AND TECHNOLOGIES (SACMAT 2002), June 2002.
25. M. Thompson, A. Essiari, S. Mudumbai, “Certificate-based Authorization Policy in a PKI Environment,” ACM Transactions on Information and System Security (TISSEC), Volume 6, Issue 4 (November 2003) pp: 566-588
26. L. Pearlman et al, “A Community Authorization Service for Group Collaboration”, 2002 IEEE Workshop on Policies for Distributed Systems and Networks
27. Lepro, R., “Cardea: Dynamic Access Control in Distributed Systems”, NASA Technical Report NAS-03-020, November 2003
28. Alfieri et al. “VOMS: an Authorization System for Virtual Organizations” 1st European Across Grids Conference, Santiago de Compostela, Feb. 13-14, 2003
29. I. Foster, C. Kesselman, J. Nick, 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.
30. K. Keahey, M. Ripeanu, and K. Doering, “Dynamic Creation and Management of Runtime Environments in the Grid”, Workshop on Designing and Building Grid Services, GGF-9, October 8, 2003, Chicago, IL
31. ITU-T Recommendation X.812, “Data Networks and Open System Communications Security”, November 1995
32. J. Vollbrecht et al., “AAA Framework”, Internet RFC2904, Internet Engineering Task Force, Network Working Group, August 2000.