Role-Based Access Control for Grid Database Services Using the Community Authorization Service

Anil L. Pereira, Vineela Muppavarapu, and Soon M. Chung, *Member*, *IEEE*

Abstract—In this paper, we propose a role-based access control (RBAC) method for Grid database services in Open Grid Services Architecture-Data Access and Integration (OGSA-DAI). OGSA-DAI is an efficient Grid-enabled middleware implementation of interfaces and services to access and control data sources and sinks. However, in OGSA-DAI, access control causes substantial administration overhead for resource providers in virtual organizations (VOs) because each of them has to manage a role-map file containing authorization information for individual Grid users. To solve this problem, we used the Community Authorization Service (CAS) provided by the Globus Toolkit to support the RBAC within the OGSA-DAI framework. The CAS grants the membership on VO roles to users. The resource providers then need to maintain only the mapping information from VO roles to local database roles in the role-map files, so that the number of entries in the role-map file is reduced dramatically. Furthermore, the resource providers control the granting of access privileges to the local roles. Thus, our access control method provides increased manageability for a large number of users and reduces day-to-day administration tasks of the resource providers, while they maintain the ultimate authority over their resources. Performance analysis shows that our method adds very little overhead to the existing security infrastructure of OGSA-DAI.

Index Terms—Open Grid Services Architecture-Data Access and Integration (OGSA-DAI), Grid database services, fine-grain authorization, Community Authorization Service (CAS), role-based access control (RBAC).

1 Introduction

Grid has emerged recently as an integration infrastructure for sharing and coordinated use of diverse resources in dynamic, distributed virtual organizations (VOs) [3], [6], [13], [14], [36], [37]. Distributed data sources can be diverse in their formats, schema, quality, access mechanisms, ownership, access policies, and capabilities. To efficiently manage these, technical solutions and standards are needed for data discovery and access, data exploration and analysis, resource management, and security [16]. To date, most work on data storage, access and transfer on the Grid has focused on files [31], but the Grid can also be used to integrate various distributed heterogeneous databases and supports query/transaction processing on them through a uniform interface [31], [32]. However, the use of databases in Grids presents different security needs and access policies compared with the use of computational resources. For example, certain applications may be authorized to access only a certain part of a database.

The Data Access and Integration Services Working Group (DAIS-WG) of the Global Grid Forum (GGF) is currently establishing the standards for Grid interface to data resources [20]. The Open Grid Services Architecture-Data Access and Integration (OGSA-DAI) provides the first implementation for these emerging standards. Currently, OGSA-DAI supports role-based access control (RBAC) [9], [30] via a role-map file that maps individual Grid users to database roles. In other words, permissions are associated with roles, and users are made members of appropriate roles, thereby acquiring the roles' permissions [29]. In this case, each resource provider has to maintain a role-map file to authorize access to its resources. This method of access control is not suitable for VOs, because both users and resources are dynamic in VOs. Multiple entries in multiple role-map files may need to be updated if new users are allowed to access multiple data resources or if the access privileges of current users change. This puts an unnecessary burden on the resource providers in managing the role-map files, especially when both the users and resource providers belong to multiple VOs.

In this paper, we describe how the security mechanism in OGSA-DAI can be enhanced by using the Community Authorization Service (CAS) provided by the Globus Toolkit [11]. The CAS records user groups and their permissions on resources and it targets access control for computational and file-based storage resources. But, we demonstrate that the CAS can also support RBAC for multiple VOs to access Grid databases within the OGSA-DAI framework. We extended the RBAC approach supported by OGSA-DAI to allow users to be assigned memberships on VO roles and also to allow role hierarchies. With our method, the CAS grants users memberships on VO roles and then authorizes them in those roles. The resource providers need to maintain only the mapping information from VO roles to local database roles, thus the number of entries in the role-map file is reduced dramatically. When users join/leave a VO, the resource providers do not have to bother about individually adding/removing

• *The authors are with the Department of Computer Science and Engineering, Wright State University, Dayton, OH 45435. E-mail: {pereira.3, muppavarapu.2, soon.chung}@wright.edu.*

Manuscript received 5 May 2005; revised 6 Feb. 2006; accepted 15 Feb. 2006; published online 4 May 2006.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0062-0505.

their information in the role-map files because the CAS server can just grant/revoke their memberships on the VO roles. Furthermore, the resource providers can grant or refuse the access requests of specific users by maintaining their authorization information separately in the role-map files. This enables the resource providers to have the ultimate authority over their resources.

We have implemented our proposed method and analyzed its performance. In our implementation, users obtain CAS credentials based on user credentials. The user credential is formed by an X.509 certificate and the associated public/private keys and is issued by a Certificate Authority (CA) trusted by all entities in a Grid [5]. The CAS credentials contain the authorization information for the user in terms of his/her VO roles. We have extended the client-side implementation of OGSA-DAI to pass the CAS credential and extended the server-side to parse the credential to obtain the VO role (in which the user is to be authorized) and to map it to a local database role. We have evaluated our solution in terms of the overheads incurred when security contexts are set up between a client and a server. This has been done with respect to the original security mechanism in OGSA-DAI.

The organization of the paper is as follows: Section 2 contains background information. In Section 3, we explain the current authorization mechanism in OGSA-DAI. In Section 4, we describe our approach to RBAC using CAS in OGSA-DAI. Section 5 describes the implementation details, and Section 6 describes the results of performance analysis. Section 7 contains some conclusions and future work.

2 Background

Grid integrates several communities of resource providers and resource consumers. This integration can be technically challenging because of the need to achieve various qualities of service when running on top of different native platforms. Open Grid Services Architecture (OGSA) addresses these challenges and defines uniform exposed service semantics, the Grid Service [15]. OGSA integrates Grid and Web services technologies and defines standard interfaces and behaviors for distributed system integration and management [16]. Version 3 of the Globus Toolkit and its accompanying Grid Security Infrastructure (GSI) provide the first implementation of OGSA mechanisms and cast security functions as OGSA services. This version of the Globus Toolkit also publishes service security policies and specifies standards for interoperability [38].

2.1 Grid Databases

Current research in the area of Grid databases is undertaken by *Project Spitfire* associated with the European Data Grid [4] and the *Open Grid Services Architecture-Data Access and Integration* (OGSA-DAI) [2]. Project Spitfire provides access control based on authorization tags specified within XML-based query files. These tags are mapped by a database resource to local roles via a role-database that it maintains. A drawback of their approach is that the role-database contains the mapping to local database roles for all Grid users that have access to that database resource. Multiple entries in multiple role-databases may need to be updated if new Grid users are allowed to access multiple data resources or if the access privileges of current users change.

OGSA-DAI is an efficient Grid-enabled middleware implementation of interfaces and services to access and control data sources and sinks [2]. In order to expose physical data resources to the Grid, by extending the interfaces defined by the Open Grid Services Infrastructure (OGSI) [35], OGSA-DAI introduced the following services [2]:

1. Grid Data Service Factory (GDSF): Represents a data resource, and exposes its capabilities and metadata.
2. Grid Data Service (GDS): Created by a GDSF and holds the client session with the data resource.
3. DAI Service Group Registry (DAISGR): Clients can discover service/data by locating GDSFs registered with a DAISGR.

2.2 Issues for Access Control in Grids

The overall direction for access control architectures in Grid computing is toward the need for leveraging IT infrastructure as it emerges. Integration with Web services and hosting environment technologies introduces opportunities to leverage emerging security standards and technologies such as the Security Assertion Markup Language (SAML) [23] and Web Services Security (WSS) [25]. Participating organizations within a Grid often have significant investment in existing security mechanisms and infrastructure, and Grid services could be built on sophisticated container-based hosting environments such as J2EE or .Net. Grid security mechanisms should interoperate with, rather than replace, those mechanisms [38].

Most security functionality should be placed in the hosting environments, so that application development will be simplified and security functionality can be upgraded independently of applications [38]. Participating organizations may have different security models. It is important for these models to interoperate based on different levels of trust. The WSS specifications address this issue. WSS is a standard mechanism for interoperability and enables the interaction between different platforms and security models. WSS standard can be used to transport credentials from a client to a server, such as the ones represented by SAML attribute assertions [33].

Resource providers must understand and support mechanisms and policies that are not strictly under their control. A VO spanning across multiple sites can use a single security mechanism, but, usually, it needs to accommodate multiple security mechanisms [17]. While acknowledging and respecting the site autonomy, there are a number of requirements to be met for Grid security in order to achieve the goals of the VOs. Users need globally defined names that are recognized at all sites they access. A user's identity needs to be passed securely and transparently between sites as jobs progress [17].

Users must be able to access resources dynamically without any administrator intervention. These resources must be coordinated properly and must interact securely with other services. Thus, resources must have global identities, and they should be accessed without violating their local policies. Trust should be established not only

among users and resources, but also among the resources themselves, so that they can be coordinated. These trust domains can span across multiple organizations and must adapt dynamically as participants join or leave and resources are accessed or released [38].

Significant challenges remain for privacy management and cross-domain auditing [17]. Also, the typical identity-based authorization used today is not scalable because authorization information should be maintained for each user. RBAC is clearly an emerging direction in Grid computing [33]. In RBAC, authorization information is associated with roles, not with individual users.

The GridShib project [39] leverages the local security infrastructures of different organizations so that users can be authenticated to Grid resources by using methods already supported at their home organizations. The goal of GridShib is to create a distributed authorization framework that supports anonymous interactions between users and, hence, protects their privacy. The rights of the users can be expressed using attributes such as institutional affiliation, group membership, or their role in collaboration [39].

GridShib incorporates the Shibboleth [8], which is an Attribute Authority service, developed by the Internet2 community for cross-organization identity federation [39]. The Shibboleth service maps a user name and attributes onto a unique identifying handle. To protect the user's privacy, the service can restrict the information about the holder of the handle depending on who is asking for the information. For example, it does not release the user's name except to those requestors who have been authorized [17].

In addition to GridShib, PERMIS [26] and the Virtual Organization Membership Service (VOMS) [1] are also attribute-based authorization services. They use assertions that bind the attributes to users for authorization, as opposed to the typical identity-based authorization used today [33]. However, currently there is no standard for how attributes are transferred from the attribute authority to the Grid services and no standard for expressing the policy regarding those attributes [33]. SAML [23] can be used to express authorization queries, and Extensible Access Control Markup Language (XACML) [24] can be used to express authorization policy statements.

An audit mechanism can be used to determine whether or not the access control policies have been administered properly. The audit mechanism is responsible for producing records which track security related events [22]. And, for this purpose, it is essential to keep a log of the access requests and the enforced security policies. In traditional systems, the audit mechanism is local to each server; however, on the Grid, either the audit mechanism should be distributed or the audit records should be transmitted to a location where a higher level view of the system can be constructed [12]. Standards are required to facilitate the audit and to reconcile different audit trails that are distributed among different organizations. It is extremely difficult to browse the audit logs if they are in different formats and in different administrative domains. Also, the access control mechanism should be able to match the audit entries in different audit logs and administrative domains.

Auditing also depends on authentication because audit records usually associate individuals with the actions they have taken, and the identity of the user must be determined if these entries are to be trusted [12]. The user identity represented in the user credential can be used to identify the user who initiated the request. The request can be logged at the resource along with the mapping information and the subsequent actions performed. This information can be used to find patterns that fit the profile of a system intrusion or the activities that do not fit the profiles of legitimate users [12].

3 Current Authorization Mechanism in OGSA-DAI

User authorization is one of the most challenging issues in Grid computing. Current authorization mechanisms cannot address all the issues that arise in dynamic Grid environments which often encompass multiple organizations, each with its own security policy [19]. RBAC shows clear advantages over traditional discretionary and mandatory access control models in such environments, because it allows the uniform representation of diverse security policies and ensures that no security violations occur during interdomain access [19].

Furthermore, RBAC is distinguished by its inherent support for the Principle of Least Privilege [21]. The Principle of Least Privilege requires that a user be given no more privileges than necessary to perform a job [9]. It can be easily enforced by first identifying the roles in an organization correctly and then assigning only those privileges to each role that allow the role members to perform their tasks. Hence, some Grid authorization mechanisms have adopted the RBAC model. With our method, users can request a particular role among those they are entitled to and, hence, gain the specific permissions tied with that role.

The current security infrastructure of OGSA-DAI uses a role-map file for authorizing a Grid user's request. The role-map file contains the information for mapping a Grid user credential to a username and a password that are used to connect to a database at a particular authorization level. Multiple entries in multiple role-map files may need to be updated if new Grid users are allowed to access multiple data resources or if the access privileges of current users change. Thus, managing the entries in a role-map file is difficult. With these considerations in mind, we propose an efficient access control mechanism for Grid database services in OGSA-DAI.

4 A Role-Based Access Control (RBAC) Method for OGSA-DAI

We enhanced the existing implementation of OGSA-DAI to use the Community Authorization Service (CAS) provided in the Globus Toolkit. CAS provides a scalable mechanism for specifying and enforcing complex and dynamic policies that govern resource usage within Grids. It allows resource providers to delegate some of the authority for maintaining

fine-grain access control policies to communities, while still maintaining the ultimate authority over their resources [27].

4.1 Community Authorization Service (CAS)

A community runs a CAS server to keep track of its membership and fine-grain access control policies. A user accessing community resources contacts the CAS server, which delegates rights to the user based on the request and the user's role within the community. These rights are in the form of capabilities, which users can present at a resource to gain access on behalf of the community. The user effectively obtains the intersection of the set of rights granted to the community by the resource provider and the set of rights defined by the capabilities granted to the user by the community. The CAS server uses a backend database to store the capabilities of the users. The CAS architecture builds on the public key authentication and delegation mechanisms provided by the Grid Security Infrastructure (GSI) [27]. The CAS server contains policy statements that specify who (which user or group) has the permission, which resource or resource group the permission is granted on, and what permission is granted [27]. The permission is denoted by a service type and an action. The action describes the operation (e.g., read, write, or execute program), and the service type defines the namespace in which the action is defined (e.g., file). Different resource providers may recognize different service types, but all resource providers that recognize the same service type should have the same interpretation of that service type's actions [27].

If a user of a community needs to gain access to a resource, the user generates a proxy credential which is signed by his/her own user credential. The proxy credential is presented to the CAS server which returns a new credential, known as a CAS proxy credential. This credential contains the CAS policy assertions representing the user's capabilities and restrictions as an extension. SAML is used as the format for the policy assertions. The CAS proxy credential is presented to the resource provider. The resource provider verifies the validity of the proxy credential, and then parses the CAS policy assertions to obtain the restrictions imposed by the CAS server. Thus, the CAS credential facilitates the mapping of the user to a local account, and the restrictions determine the operations the user is allowed to perform.

4.2 Advantages of Using CAS

With the CAS structure and RBAC, our method provides scalability in terms of the number of users and VOs. The CAS structure reduces the number of necessary trust relationships from $C \times P$ to $C + P$, when there are C consumers and P providers. Each consumer needs to be known and trusted by the CAS server, but not by each provider. Similarly, each provider needs to be known and trusted by the CAS server, but not by each consumer [27]. A single CAS server can support the authorization for multiple VOs. Also, it has been shown that the cost of administering RBAC is proportional to $U + P$ per role, while the cost of associating users directly with permissions is proportional to $U \times P$, where U is the number of individuals in a role and P is the number of permissions required by the role [10], [40].

However, in terms of the actual number of access requests on resources, using a single CAS server may not be quite scalable. A single CAS server can be a bottleneck if a large number of users attempt to access it at the same time, and it can be a single point of failure. A possible solution for these problems depends on how frequently the community policies change. If the community policies do not change frequently, a single master server can be maintained to accept the changes and then routinely replicate the policies to one or more read-only slave servers. If the community policies change frequently, multiple peer servers can be used. All the servers update the policies, so that the failure of any one server will not lead to a loss of functionality [27].

However, when policies are changing dynamically, the complete centralization of policies can achieve better consistency. Also, in the case that a user credential is compromised, revocation is easier when a single CAS server is used because the user needs to be removed only from that server [27].

CAS comes packaged within the Globus Toolkit and is easily deployable. CAS also has advantages over the existing authorization services that support RBAC within Grids. Among these are Virtual Organization Management Service (VOMS) [1] and Akenti [34]. Compared with VOMS, CAS assertions provide the rights directly and do not need an interpretation by the resource. Even though CAS was designed primarily for fine-grain policies, it also has been shown to be capable of asserting coarse-grain group memberships [7], [28]. As far as Akenti is concerned, it is targeted on authorizing accesses to web resources and particularly Websites, so it is not adequate for VOs [1]. Akenti does not provide support for dynamic delegation [26]. Delegation is a key issue in a VO, wherein a set of rights can be delegated to a program for it to act on behalf of a user. A program should also be able to delegate some of its rights to other programs [13].

4.3 Role-Based Access Control with CAS in OGSA-DAI

A possible approach for supporting RBAC with CAS is the use of rights associated with a role to access role-specific resources [7]. The role of a user is presented in a hierarchical form. For example, Alpha/admin indicates the administrator role of a virtual organization Alpha. Alpha could be the name of a project undertaken by collaborating organizations.

Due to the current implementation of CAS, a drawback of this approach is the possibility that, while a user acts in a role which does not possess access privileges for a particular resource, he/she could be authorized by CAS to access that resource. As an example, suppose that users of userGroup1 have the membership right on a role "Alpha/programmer" and also have the read right on "ftp://localhost/tmp/fileA.txt," and the users of userGroup2 have the membership right on a role "Alpha/guest" and also have the read right on "ftp://localhost/tmp/fileB.txt." If a user "user1" is in both userGroup1 and userGroup2, as illustrated in Figs. 1 and 2, CAS will authorize him/her in the role "Alpha/programmer" together with the read access on "ftp://localhost/tmp/fileB.txt," but it is not a valid access right for that role.

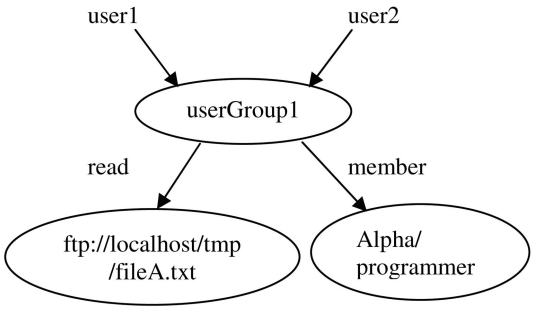

Fig. 1. UserGroup1 with a role Alpha/programmer and read access to ftp://localhost/tmp/fileA.txt.

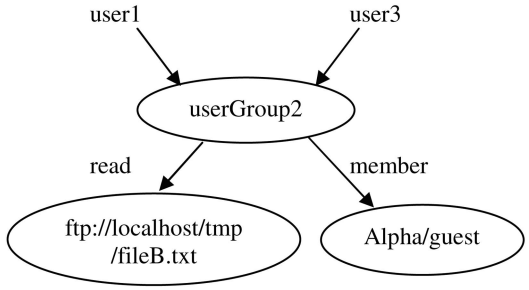

Fig. 2. UserGroup2 with a role Alpha/guest and read access to ftp://localhost/tmp/fileB.txt.

Our implementation with CAS is based on the following idea proposed in [7]: In the CAS database, roles are added as resources and users are given membership rights on those roles. In a VO with a large number of users, we could think of several groups of users, each with different levels of access (roles). A role has certain privileges associated with it. When a VO role is mapped to a local role, it will specify the access a user can have; for example, a specific table of a database. For mapping, the resource provider can obtain the policy details through the user's CAS credential and interpret it to a known level. A role can be assigned to any number of users. When users join/leave the VO, the resource provider does not have to bother about individually adding/removing them from the role-map files because the CAS server could just grant/revoke their membership from the existing VO roles.

In our approach, CAS grants the membership rights on roles to users by assigning them to appropriate user groups in the CAS database. However, we do not associate the rights with roles to access role-specific resources. Instead, the decision to map a VO role to a local database role and the assignment of fine-grain privileges to the local role is the responsibility of the resource provider. The fine-grain privileges and constraints associated with the local role can be negotiated between the VO and the resource provider. But, the resource provider has the control over the actual assignment of fine-grain privileges to the local role and the specification of constraints on it. For example, a resource provider can grant permission to perform basic database operations (e.g., select) on a particular database table. The resource provider can also grant permissions for more complex operations such as executing stored database procedures.

Furthermore, the resource provider can grant or refuse the access requests of specific users by maintaining their authorization information separately in the role-map files. This enables the resource provider to have the ultimate authority over its resources. With our method, a user can delegate a subset of his/her authorized VO roles to certain applications and Grid Data Services. In this case, the privileges associated with the delegated VO roles are the privileges associated with the corresponding local roles.

As shown in Fig. 3, a resource provider can decide to map the Alpha/supervisor role to a local role that allows the function viewInventory( ) to be performed between 19:00 and 5:00 GMT from Monday to Friday during 05.20.2005-07.30.2005. This specified timing constraint can be enforced by a database trigger, which executes an action automatically on the occurrence of a predefined event.

5 Implementation Details

Our RBAC method with CAS supports the push model, where the user directly obtains the permissions from the authorization server and passes them to the target resources at the time of making a request. The resource verifies the authenticity of the user and then authorizes the user based on the permissions obtained, provided the authority that issued them is trustworthy. Our decision to use the push

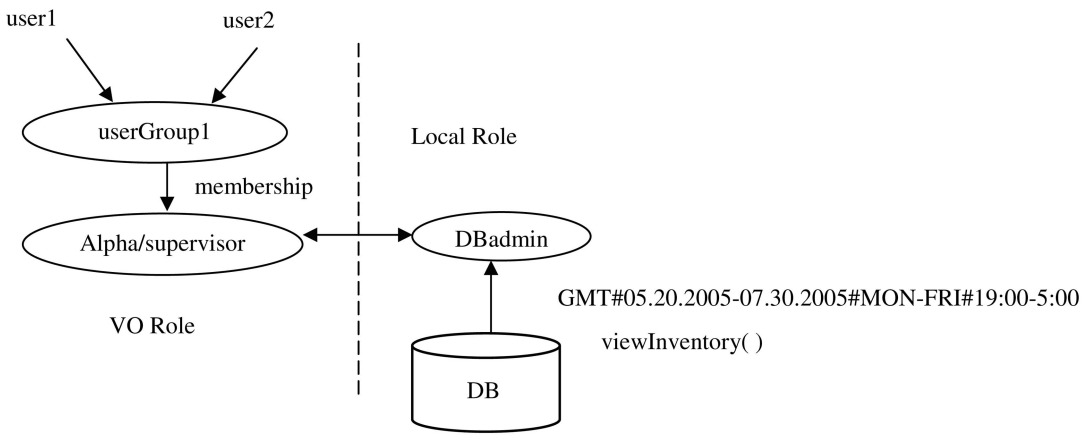

Fig. 3. Specifying VO roles using CAS.

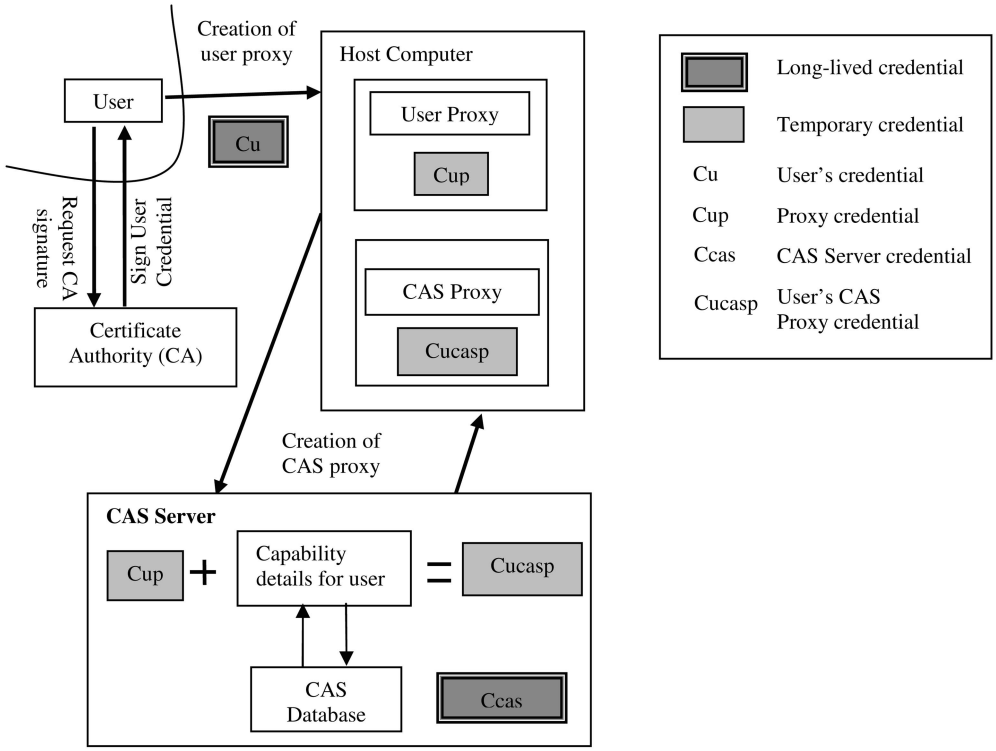

Fig. 4. User's normal proxy credential and CAS proxy credential creation.

model is based on the advantage that the user can explicitly select a role. Also, in the case that the user and the authorization service belong to the same organization and are protected by a firewall, the authorization service using the push model should be deployed because the resources may not be able to contact the authorization service directly.

Some authorization services, like Akenti, support the pull model, where the user is authenticated by a target resource. The target resource contacts the authorization server to obtain the user's permissions. An advantage of the pull model is that it can be deployed easily because users do not need to interact with the authorization service [39]. We plan to enhance our method to support the pull model as well so that the OGSA-DAI services can contact CAS and obtain the user's permissions directly. Supporting both models will allow flexible deployment to meet the needs of different VOs.

CAS has a backend database for storing information about users, resources and associated privileges. The VO members are granted user credentials signed by a Certificate Authority (CA). CAS issues a certificate to authorize users based on their requested role, their user credentials, and the role membership information in the CAS database. The CAS database administrator can delegate the right to grant/revoke memberships on roles to other users, and those users can exercise that right only within the user groups to which they belong.

CAS provides a set of APIs for managing fine-grain access policies for resources in a VO [27]. The Service API of CAS provides an administrative interface for managing the user groups and associated privileges. This API supports the user's role assignments in our method. CAS also provides a Client API through which users can obtain a signed SAML assertion and present it to the resource provider for authorization. The OGSA-DAI client program uses the Java Generic Security Services API (GSSAPI) to delegate the CAS credential to a Grid Data Service (GDS).

We configured CAS to incorporate the proposed RBAC method as described before and modified the OGSA-DAI implementation to make use of the CAS credentials. The modifications are made at both client-side and server-side. The client is modified to delegate the CAS credential instead of the user proxy credential. The server is modified to recognize the CAS credential delegated by the client, obtain the role from it using the GSSAPI libraries, and perform the role mapping based on the role-map file. The role-map file has been extended to include the mapping from a VO role to a database username and a password.

The following is the sequential process of a user obtaining a CAS credential and accessing a Grid database. As shown in Fig. 4, a user generates a certificate by making a request to a Certificate Authority (CA) which is also trusted by all the entities within the Grid, i.e., all users and resources. Once the user has obtained the certificate (Cu), a proxy credential (Cup) is generated based on the certificate. This generated proxy credential's life time will be less than the lifetime of the user certificate. The lifetime of a proxy credential generated using the Globus Toolkit is 12 hours.

Normally, whenever the user wants to access a resource, the proxy credential can be used. The resource provider checks the validity of the user's proxy credential, authenticates the user, and maps the user to a local account. In order to use a CAS credential, the user initiates a CAS proxy by making a request to the CAS server based on the user's proxy credential. The CAS server authenticates the user and obtains the user's capabilities (in the form of roles) present

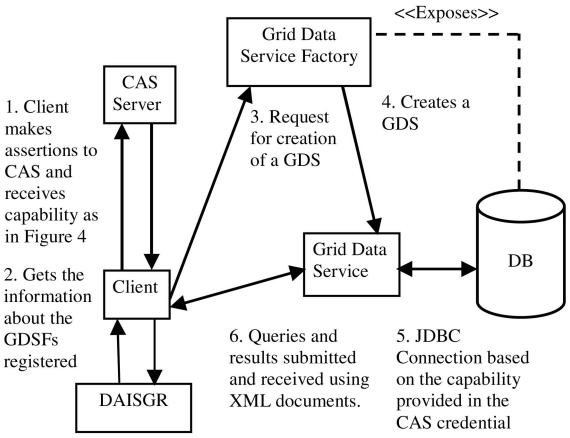

Fig. 5. Accessing a data resource through OGSA-DAI using a CAS credential.

in the CAS database. The CAS server then creates a new proxy credential (Cucasp) by adding these capabilities to the existing user proxy credential (Cup).

As shown in Fig. 5, once the user has obtained the CAS credential with the requested assertions, the user can contact the desired Grid Data Service Factory (GDSF) to create a Grid Data Service (GDS). The GDS gets the CAS credential delegated by the user and sets the security context with the role present in the CAS policy assertions.

As shown in Fig. 6, the user first initiates a user proxy and then contacts CAS to initiate the CAS proxy. While making a request to create a CAS proxy credential, the required role can be explicitly specified by using a file containing user specific requests. Following from the previous example that describes the mapping of the Alpha/supervisor role to a local role, a user makes a request for the Alpha/supervisor role as shown in Fig. 6.

Fig. 7 shows the SAML representation of a part of the CAS credential that contains the role Alpha/supervisor. The SAML representation of the credential can be seen using Globus/CAS command-line tools. Based on the role, a JDBC connection is established between the Grid Data Service and the database exposed by the Grid Data Service Factory. If no role is specified in the CAS credential, then the user's identity is used for mapping. The client can then

```
<AuthorizationDecisionStatement
Decision="permit"
Resource="roleNamespace|Alpha/supervisor">
<Subject>……</Subject>
<Action
Namespace="group">membership</Action>
</AuthorizationDecisionStatement>
```

Fig. 7. Authorization information present in the CAS credential specifying user's VO role.

submit queries to the Grid Data Service and obtain the results in XML documents as shown in Fig. 6.

6 Performance Analysis

The existing implementation of the OGSA-DAI client has been modified to delegate a CAS credential, and the server has been modified to obtain the roles present in the CAS credential. The overheads incurred with our implementation are compared with those of the existing implementation of OGSA-DAI, which does not use the CAS credential. OGSA-DAI Release 4.0 was deployed on a Jakarta Tomcat 5.0.27/Globus Toolkit 3.2.1 (GT3) stack running on a Linux machine with a 2.6 GHz Intel Pentium IV processor and 1 GB of RAM. The *littleblackbook* MySQL database table distributed with OGSA-DAI was used as a test database, and it contains 10,000 tuples. The *perform* document consisting of a request for a single tuple was used for the purpose of analysis.

6.1 Profiling Details

A Java method *System.currentTimeMillis*() is used to get the current system time in milliseconds. Also, for the server-side analysis, the Apache Log4j logger, which logs time to a log file in milliseconds, is used. For more accuracy, the tomcat container was shut down and restarted before each client request in order to minimize the caching effects within GT3 and OGSA-DAI [18]. The main changes from the original configuration are the way the mapping is done at the server-side and how the credential is delegated at the client-side. So, only the security aspects of the client and the server are profiled and analyzed. The following types of Grid Data Services are used in the analysis as in [18]:

```
# Initiate a User Proxy
% grid-proxy-init
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-motive.cs.wright.edu/OU=cs.wright.edu/CN=Vineela Muppavarapu
Enter GRID pass phrase for this identity:
Creating proxy... Done
Your proxy is valid until: Wed Oct 5 20:30:53 2005

#Initiate a CAS Proxy
%cas-proxy-init -c http://localhost:8080/ogsa/services/base/cas/CASService -f /home/vinny3k/roleRequest -t tag
#File "roleRequest' with user specific request for CAS credential used above
Resource: roleNamespace|Alpha/supervisor group membership

#Contacting a specific GDSF using CAS capabilities
%java uk.org.ogsadai.client.Client -mls -t tag -factory
http://130.108.17.176:8080/ogsa/services/ogsadai/SecureGridDataServiceFactory
examples/GDSPerform/JDBC/query/select1Row.xml
```

Fig. 6. User session accessing a GDS using CAS.

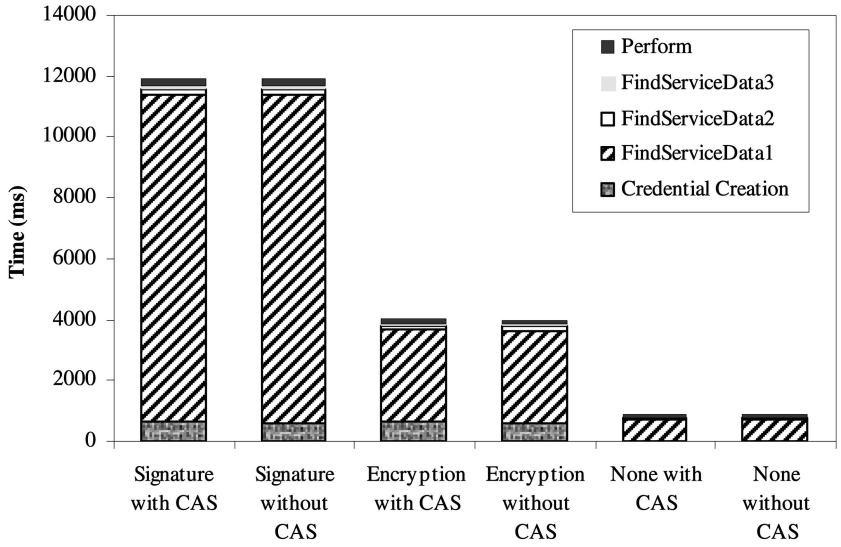

Fig. 8. Client-side security.

1. *Signature:* GDS enforcing GSI Secure Conversation with Signature. This enforces message integrity being established between the client and the server.
2. *Encryption:* GDS enforcing GSI Secure Conversation with Encryption. This enforces message privacy being established.
3. *None:* GDS which does not enforce any security. The GDS does not provide a secure conversation.

6.2 Client-Side Security

A call is made to each of the above Grid Data Services using a CAS proxy credential and without using a CAS proxy credential. In case of using a CAS proxy credential, an additional overhead for creating the proxy credential is incurred. The lifetime of the CAS proxy credential is equal to the time remaining for the expiration of the user proxy credential, which can last up to 12 hours. However, this overhead is incurred only once before establishing the security context initially. Thereafter, the client can make any number of queries before the proxy credential expires.

The *findServiceData* method of a GDSF returns the information about its corresponding data resource. Three consecutive calls to *findServiceData* are required: The first call returns the database schema, the second returns the activities permitted, and the third returns the product type (for example, the type of DBMS, say MySQL). The *perform* method of a GDS takes the *perform* document, which contains the query, and returns the results to the client.

GSI Secure Conversation requires a security context to be established between the client and the server. The overheads incurred in setting up this security context are analyzed based on the following:

1. Calls made for creating a credential object from the proxy credential.
2. Calls to the *findServiceData* and *perform* methods.

The corresponding time durations are shown in Fig. 8, and, as observed, the time for creating the credential object is almost the same regardless of the security enforced by the GDS. In case of *None*, there is no such overhead as the credentials are not used. The first call to the *findServiceData* takes longer than the subsequent calls because it includes the initialization of the GDS regardless of the security type used.

The times recorded in the case of using a CAS proxy credential and those without using a CAS proxy credential are almost the same. The reason is that all the security functions on the client-side remain unchanged except for the use of a CAS proxy credential instead of a user proxy credential.

6.3 Server-Side Security

The analysis made on the server-side is based on the following:

1. The client credentials accessed using the GT3 infrastructure.
2. Extracting the VO role or Grid identity from the credential.
3. Mapping a user to a database username and a password, and creating a JDBC connection.
4. The *perform* operation.

As shown in Fig. 9, the time for executing the *perform* operation remains constant for all the GDSs. It also shows the time for the credential extraction is very small compared to the time for executing the *perform* operation. In Fig. 10, the credential extraction times are shown clearly, and we can see that the credential extraction takes more time when a CAS proxy credential is used for contacting a GDS that enforces the secure conversation. This is because of the time taken to obtain the role from the CAS credential. In case of contacting a nonsecure GDS, there is no difference as no credential is used.

Fig. 11 shows that there is a constant overhead for mapping a user to a database username and a password and then subsequently setting up the database connection.

7 Conclusion

In this paper, we enhanced the role-based access control (RBAC) mechanism of OGSA-DAI by using the Community

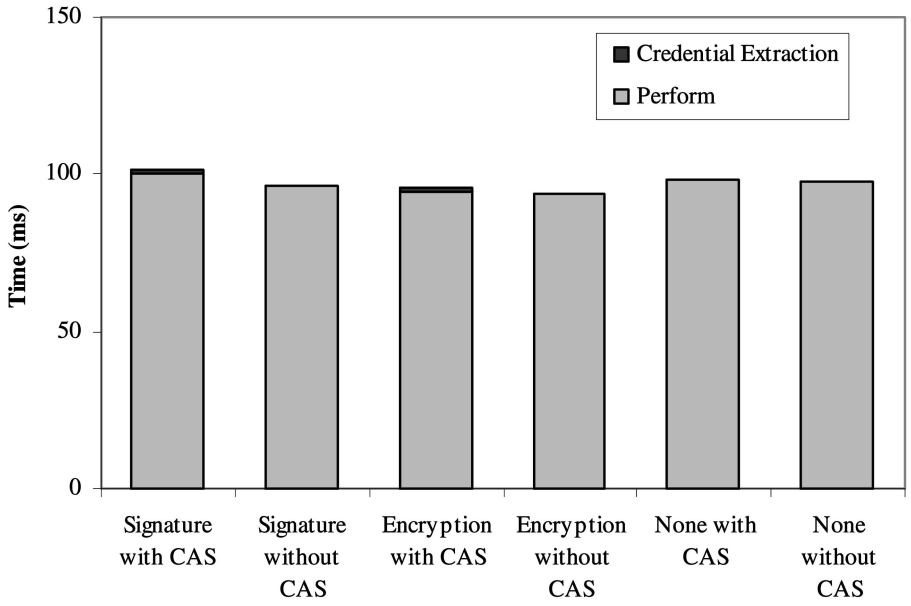

Fig. 9. Server-side security.

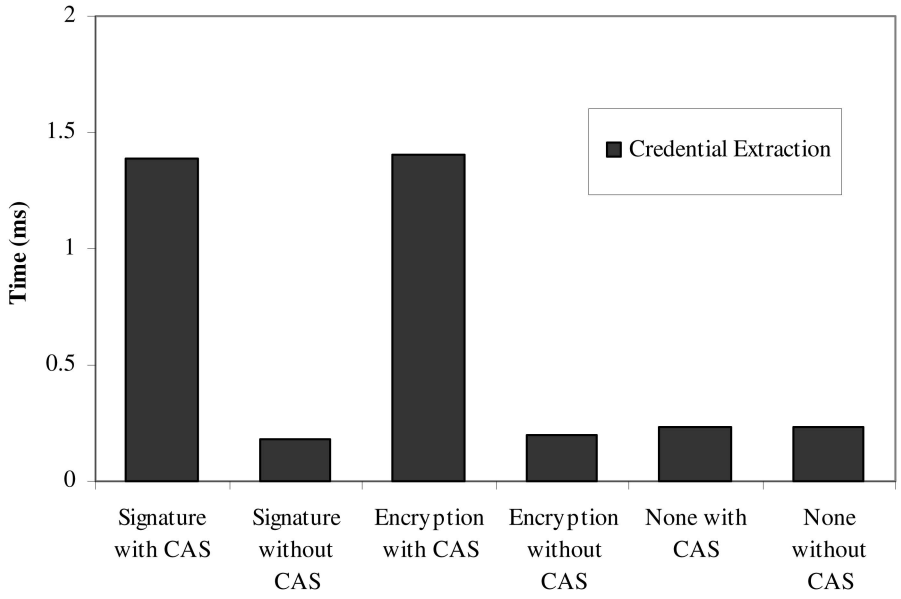

Fig. 10. Security overheads on the server-side.

Authorization Service (CAS) so that users are granted memberships on virtual organization (VO) roles for Grid database services. The resource providers need to maintain only the mapping information from VO roles to local database roles; thus, the number of entries to be managed in the role-map file is reduced dramatically. When users join or leave a VO, the resource providers do not need to add or remove their information individually in the role-map files because the CAS server can just grant or revoke their memberships on VO roles. Furthermore, the resource providers can grant or refuse the access requests of specific users by maintaining their authorization information separately in the role-map files. This enables the resource providers to have the ultimate authority over their resources. Our performance analysis shows that the proposed RBAC method using CAS provides a scalable means of access control for databases in the Grid. Without taking much extra time to set up the security context between the client and the server, CAS brings significant advantages to the authorization mechanism of OGSA-DAI in terms of the manageability for a large number of users and reduced administration overheads.

In our future work, we will design and implement an enhanced RBAC model for Data Grids supported by CAS. Our current approach uses CAS to specify role memberships only and the CAS assertions are interpreted by the resource provider. Our method can be enhanced to have the CAS assertions include both role memberships and associated privileges so that the resource providers do not need to interpret the CAS assertions. Furthermore, CAS can also

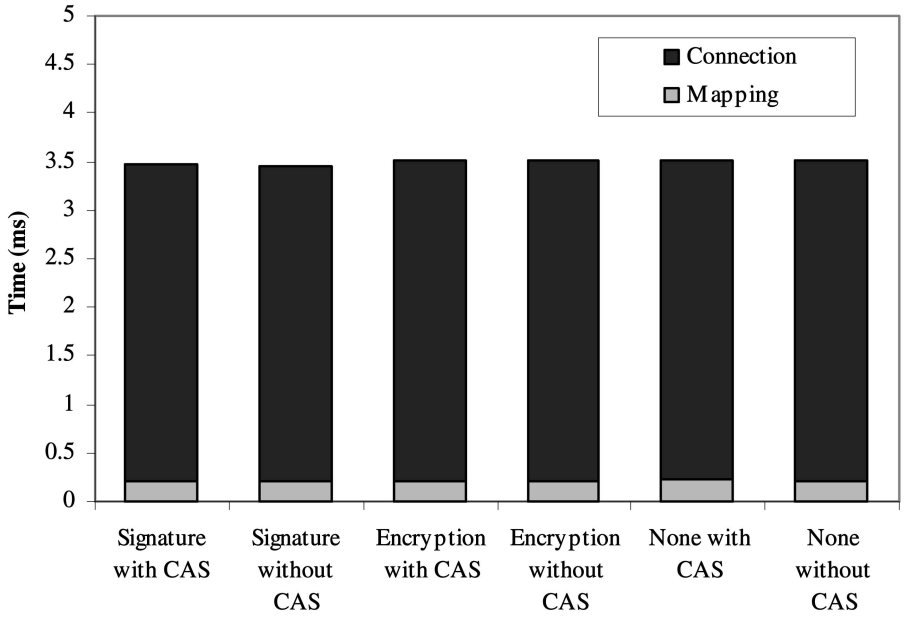

Fig. 11. Mapping and database connection.

be used to specify constraints on a role by defining the service types and actions appropriately.

While our method provides security in terms of access control, it does not provide privacy protection for the users because every CAS credential contains information that identifies the user. We will investigate various privacy protection mechanisms in the Grid. For example, to protect the user's identity, a pseudonym identity can be used in the CAS credentials.

ACKNOWLEDGMENTS

This research was supported in part by AFRL/Wright Brothers Institute (WBI).

REFERENCES

[1] R. Alfieri et al., "Managing Dynamic User Communities in a Grid of Autonomous Resources," *Proc. Int'l Conf. Computing in High Energy and Nuclear Physics,* 2003.

[2] A. Anjomshoaa et al., "The Design and Implementation of Grid Database Services in OGSA-DAI," *Proc. UK e-Science All Hands Meeting,* 2003.

[3] A.E. Arenas et al., "Toward Web Services Profiles for Trust and Security in Virtual Organizations," *Proc. Sixth IFIP Working Conf. Virtual Enterprises,* pp. 26-28, 2005.

[4] W.H. Bell, D. Bosio, W. Hoschek, P. Kunszt, G. McCance, and M. Silander, "Project Spitfire—Towards Grid Web Service Databases," informational document, Global Grid Forum, 2002.

[5] R. Butler, V. Welch, D. Engert, I. Foster, S. Tuecke, J. Volmer, and C. Kesselman, "A National-Scale Authentication Infrastructure," *Computer,* vol. 33, no. 12, pp. 60-66, Dec. 2000.

[6] L.M. Camarinha-Matos and H. Afsarmanesh, "A Roadmap for Strategic Research on Virtual Organizations," *Proc. Fourth IFIP Working Conf. Virtual Enterprises,* pp. 33-46, 2003.

[7] S. Cannon, S. Chan, D. Olson, C. Tull, V. Welch, and L. Pearlman, "Using CAS to Manage Role-Based VO Sub-Groups," *Proc. Int'l Conf. Computing in High Energy and Nuclear Physics,* 2003.

[8] S. Carmody, "Shibboleth Overview and Requirements," Shibbololeth Working Group Document, http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-requirements-01.html, 2001.

[9] D. Ferraiolo and R. Kuhn, "Role-Based Access Control," *Proc. 15th Nat'l Computer Security Conf.,* 1992.

[10] D.F. Ferraiolo, J.F. Barkley, and D.R. Kuhn, "A Role-Based Access Control Model and Reference Implementation within a Corporate Intranet," *ACM Trans. Information and System Security,* vol. 2, no. 1, pp. 34-64, 1999.

[11] I. Foster and C. Kesselman, "The Globus Toolkit," *The Grid: Blueprint for a New Computing Infrastructure,* I. Foster, C. Kesselman, eds., pp. 259-278, Morgan Kaufmann, 1999.

[12] I. Foster and C. Kesselman, "Security, Accounting, and Assurance," *The Grid: Blueprint for a New Computing Infrastructure,* I. Foster and C. Kesselman, eds. pp. 395-420, Morgan Kaufmann, 1999.

[13] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. Supercomputer Applications and High-Performance Computing,* vol. 15, no. 3, pp. 200-222, 2001.

[14] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke, "Grid Services for Distributed System Integration," *Computer,* vol. 35, no. 6, pp. 37-46, June 2002.

[15] I. Foster, C. Kesselman, J., M. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure Working Group, Global Grid Forum, 2002.

[16] I. Foster and R.L. Grossman, "Data Integration in a Bandwidth-Rich World," *Comm. ACM,* vol. 46, no. 11, pp. 50-57, 2003.

[17] M. Humphrey, M.R. Thompson, and K.R. Jackson, "Security for Grids," *Proc. IEEE,* vol. 93, no. 3, pp. 644-652, 2005.

[18] M. Jackson, M. Antonioletti, N.C. Hong, A. Hume, A. Krause, T. Sugden, and M. Westhead, "Performance Analysis of the OGSA-DAI Software," *Proc. UK e-Science All Hands Meeting,* 2004.

[19] J.B.D. Joshi, R. Bhatti, E. Bertino, and A. Ghafoor, "Access-Control Language for Multidomain Environments," *IEEE Internet Computing,* vol. 8, no. 6, pp. 40-50, Nov.-Dec. 2004.

[20] S. Malaika, A. Eisenberg, and J. Melton, "Standards for Databases on the Grid," *ACM SIGMOD Record,* vol. 32, no. 3, pp. 92-100, 2003.

[21] T. Mayfield, J.E. Roskos, S.R. Welke, and J.M. Boone, "Integrity in Automated Information Systems," technical report, Nat'l Computer Security Center, 1991.

[22] N. Nagaratnam, P. Janson, J. Dayka, A. Nadalin, F. Siebenlist, V. Welch, I. Foster, and S. Tuecke, "The Security Architecture for Open Grid Services," Open Grid Service Architecture Security Working Group, Global Grid Forum, 2002.

[23] *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) Version 1.1,* Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, 2003.

[24] *Extensible Access Control Markup Language (XACML) Version 1.0,* Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/committees/xacml, 2003.
[25] *Web Services Security: SOAP Message Security Version 1.0,* Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, 2004.
[26] S. Otenko and D. Chadwick, "A Comparison of the Akenti and PERMIS Authorization Infrastructures," http://sec.isi.salford.ac.uk/download/AkentiPERMISDeskComparison2-1.pdf, 2003.
[27] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A Community Authorization Service for Group Collaboration," *Proc. Third IEEE Int'l Workshop Policies for Distributed Systems and Networks,* 2002.
[28] L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke, "The Community Authorization Service: Status and Future," *Proc. Int'l Conf. Computing in High Energy and Nuclear Physics,* 2003.
[29] C. Ramaswamy and R.S. Sandhu, "Role-Based Access Control Features in Commercial Database Management Systems," *Proc. 21st Nat'l Information Systems Security Conf.,* 1998.
[30] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, "Role-Based Access Control Models," *Computer,* vol. 29, no. 2, pp. 38-47, Feb. 1996.
[31] J. Smith et al., "Distributed Query Processing on the Grid," *Int'l J. High Performance Computing Applications,* vol. 17, no. 4, pp. 353-367, 2003.
[32] H. Stockinger, "Distributed Database Management Systems and the Data Grid," *Proc. 18th IEEE Symp. Mass Storage Systems and the Ninth NASA Goddard Conf. Mass Storage Systems and Technologies,* 2001.
[33] *Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective,* The Globus Security Team, http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf, 2005.
[34] M.R. Thompson, A. Essiari, K. Keahey, V. Welch, S. Lang, and B. Liu, "Fine-Grained Authorization for Job and Resource Management Using Akenti and the Globus Toolkit," *Proc. Int'l Conf. Computing in High Energy and Nuclear Physics,* 2003.
[35] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, and P. Vanderbilt, *Grid Service Specification, Draft 4,* Open Grid Service Infrastructure Working Group, Global Grid Forum, 2002.
[36] G. Wasson and M. Humphrey, "Policy and Enforcement in Virtual Organizations," *Proc. Fourth Int'l Workshop Grid Computing,* pp. 125-132, 2003.
[37] G. Wasson and M. Humphrey, "Towards Explicit Policy Management for Virtual Organizations," *Proc. Fourth IEEE Int'l Workshop Policies for Distributed Systems and Networks,* pp. 173-182, 2003.
[38] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, "Security for Grid Services," *Proc. 12th Int'l Symp. High-Performance Distributed Computing,* pp. 48-57, 2003.
[39] V. Welch, T. Barton, K. Keahey, and F. Siebenlist, "Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration," *Proc. Fourth Ann. Public Key Infrastructure R&D Workshop,* 2005.
[40] G. Zhang and M. Parasher, "Dynamic Context-Aware Access Control for Grid Applications," *Proc. Fourth Int'l Workshop Grid Computing,* pp. 101-108, 2003.

Anil L. Pereira received the BE degree in electronics engineering from Bombay University, India, in 1999 and the MS degree in computer science from Wright State University, Dayton, Ohio, in 2002. He is a doctoral student in the Department of Computer Science and Engineering at Wright State University, Dayton, Ohio. His current research interests include Grid computing, databases, and XML.

Vineela Muppavarapu received the bachelor's degree in information science and technology in 2003 from Nagarjuna University, India, and the master's degree in computer science in 2005 from Wright State University. She is currently a PhD candidate in the department of Computer Science and Engineering at Wright State University. Her current research areas include Grid computing and databases.

Soon M. Chung received the BS degree in electronic engineering from Seoul National University, Korea, in 1979, an MS degree in electrical engineering from Korea Advanced Institute of Science and Technology in 1981, and a PhD degree in computer engineering from Syracuse University, New York, in 1990. He is currently a professor in the department of Computer Science and Engineering at Wright State University. His current research interests include database, data mining, Grid computing, text mining, XML, and parallel and distributed processing. He is a member of the IEEE.