2009

Semantic and Role-Based Access Control for Data Grid Systems

Vineela Muppavarapu
Wright State University

SEMANTIC AND ROLE-BASED ACCESS CONTROL FOR DATA GRID SYSTEMS

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

By

VINEELA MUPPAVARAPU
M. S., Wright State University, 2005
B. Tech., Nagarjuna University, India, 2003

2009
Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

11/17/2009

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Vineela Muppavarapu ENTITLED Semantic and Role-based Access Control for Data Grid Systems BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy.

_____
Soon M. Chung, Ph.D.
Dissertation Director

_____
Arthur Ardeshir Goshtasby, Ph.D.
Director, Computer Science and Engineering
Ph.D. Program

_____
Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

Committee on Final Examination

_____
Soon M. Chung, Ph.D.

_____
Nikolaos Bourbakis, Ph.D.

_____
Yong Pei, Ph.D.

_____
Xinhui Zhang, Ph.D.

_____
Michael Talbert, Ph.D.
USAF AFRL

ABSTRACT

Muppavarapu, Vineela. Ph. D., Department of Computer Science and Engineering, Wright State University, 2009. Semantic and Role-based Access Control for Data Grid Systems.

A Grid is an integration infrastructure for sharing and coordinated use of diverse resources in dynamic, distributed virtual organizations (VOs). A Data Grid is an architecture for the access, exchange, and sharing of data in the Grid environment. Distributed data resources can be diverse in their formats, schema, quality, access mechanisms, ownership, access policies, and capabilities. In recent years, several organizations have started utilizing Grid technologies to deploy data-intensive and/or computation-intensive applications. As more and more organizations are sharing data resources and participating in Data Grids, the complexity and heterogeneity of the systems is increasing constantly, but their management techniques are not evolving making the systems more complicated and error-prone, indicating a clear need for standardized mechanisms to manage access control for the shared data resources.

The Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) and the Storage Resource Broker (SRB) are widely used frameworks for the integration of heterogeneous data resources in Data Grid systems. However, in these systems, access control causes substantial administration overhead for the resource providers because the authorization information has to be maintained for individual Grid users. In addition, access control policies need to specified and managed across multiple organizations. And, each organization in a Data Grid may use its own terminology to describe a resource making it difficult to coordinate between the organizations.

This dissertation focuses on solving these problems and provides access control systems that are based on existing standards. We developed a role-based access control (RBAC) system with Shibboleth, which is an attribute authorization service currently being used in many Grid applications. We used the Core and Hierarchical RBAC profile of the eXtensible Access Control Markup Language (XACML) standard for specifying access control policies uniformly across different organizations. For distributed administration of those policies, we used the Object, Metadata and Artifacts Registry (OMAR). OMAR is based on the e-business eXtensible Markup Language (ebXML) registry specifications developed to achieve interoperable registries and repositories.

We developed a semantic-based access control method using the ontology to resolve the semantic differences in terminologies. Understanding the semantics of the data being protected is often helpful in determining which users can access the data and what access level the users can have. Web Ontology Language (OWL) is used to represent the ontology of the data resources and users. By using ontology, VOs can resolve the differences in their terminologies and specify access control policies based on concepts and user roles, instead of individual data resources and user identities.

Administration of XACML policies is a difficult task because each XACML policy has several components, and the number of XACML policies may be very large in a Data Grid environment. However, no efficient tool is available for the creation and update of XACML policies. So, we developed an XACML administration tool and a GUI in Java. The tool allows the creation of XACML policies from existing RBAC policies. The tool also provides capabilities to update or create new RBAC policies. Using this tool, the

policy administrator can create new users, roles, data resources, and actions. It allows the administrator to change the user-role assignment and the permissions on a role.

Our proposed access control systems allow quick and easy deployments, and privacy protection. The systems are scalable, and support interoperability and fine-grain access control. Administration overheads for the resource providers are reduced because they do not need to maintain the individual user information. Moreover, our system allows unauthorized requests to be denied before establishing a connection to the resource, thereby reducing the connection overheads and making the data resources to be available to authorized users. Performance analysis shows that our systems add very little overhead to the existing security infrastructures of SRB and OGSA-DAI.

TABLE OF CONTENTS

LIST OF FIGURES

DEDICATION

Dedicated to my loving grandmother
Late Mrs. Seetaravamma Muppavarapu.

ACKNOWLEDGEMENT

I would like to thank my advisor, Professor Soon Chung, for his guidance and most importantly for spending his valuable time and expertise to improve my work. His encouragement and valuable suggestions are not just limited to the dissertation but also related to career. Thank you very much Dr. Chung for everything you have done for me.

I would like to thank all my committee members, Dr. Nikolaos Bourbakis, Dr. Yong Pei, Dr. Xinhui Zhang, and Dr. Michael Talbert, for their time and suggestions. I especially thank Dr. Bourbakis, Dr. Pei, Dr. Ronald Taylor and Mr. Michael Ondrasek for their support during my career search. I would also like to thank the Dayton Area Graduate Studies Institute (DAGSI) and the Department of Computer Science and Engineering for supporting my research.

I also thank my senior and colleague Anil L. Pereira with whom I have co-authored my first journal paper and few more papers after that. Additionally, I would like to thank all the members of the Database Research Laboratory with whom I have interacted during my research. I thank all my seniors, and friends, especially Sanjay, Manoj, and Pratap, and my roommates and who have given me tremendous support and encouragement during all my graduate studies.

I would like to thank all my family members for their love, concern and support. I would also like to thank all my relatives, and my cousins, especially Anil, Sushma and her husband Sai, for their constant support and encouragement. Finally, I would like to thank my loving husband Bharat for his understanding and support.

1. INTRODUCTION

In recent years, the amount of data being stored, accessed and analyzed has increased tremendously. The advancements in hardware and software have further facilitated the storage and access of this enormous data. In addition, the datasets are also stored in a distributed manner and are shared by many people across several different domains. Grid technologies and infrastructure are developed to support the sharing and coordinated use of diverse resources in dynamic, distributed virtual organizations (VOs) [20]. A virtual organization is termed as a set of individuals and resources across multiple organizations (for example, a collaboration formed by two independent physical organizations). These individuals and resources can be dynamic in nature.

A Data Grid facilitates the coordination and sharing of a large number of geographically distributed heterogeneous datasets across different domains. Data Grid provides a distributed system middleware that allows different communities to access and share data, networks, and other resources in a controlled and secure manner [21]. It also provides services for data storage, access and transfer. Data Grids support data-intensive and computation-intensive applications. Data Grids support replica and metadata management. The Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) and the Storage Resource Broker (SRB) are widely used frameworks for

the integration of heterogeneous data resources in Grid systems. OGSA-DAI is a widely used middleware infrastructure that facilitates uniform access to data resources using a service-oriented architecture (SOA) [3]. Storage Resource Broker (SRB) [5, 64], developed by San Diego Supercomputing Center (SDSC), is a data grid management system that provides applications a uniform Application Programming Interface (API) to access heterogeneous distributed storage resources.

1.1. Motivation and Problem Statement

Most of the access control mechanisms currently being used in Data Grids are identity-based, and are not scalable for the increasing number of users and datasets. As users and data resources join/leave the organizations frequently updating the access control information on an individual basis would be a difficult task. As people and policies change frequently there is no standard way of exchanging the diverse policies across several different domains. Also, all the domains might not have the same security policy and hence, there should be support for the diverse security policies.

Managing a common terminology and a common access control policy for the diverse organizations involved in the Data Grid is not feasible. Thus, making it difficult for the organizations to coordinate and share the data resources. Organizations intending to share their data resources have to deal with these differences and make the data resources available to the authorized personnel. Creating and updating access control policies for all the data resources in the Data Grid, while supporting the diverse terminologies, is a difficult task.

In addition, a user might not want to reveal the personal information; hence, privacy protection becomes an important issue. The current identity-based access control mechanisms do not protect the privacy of the user. Thus, in Data Grids the access control mechanism should not only be scalable but should also support the diverse security policies for distributed systems, and should protect the privacy of the users. This dissertation focuses on addressing these problems and provides a scalable access control mechanism for accessing multiple heterogeneous datasets distributed across diverse domains.

1.2. Our Previous Work

In [60], we described how the Community Authorization Service (CAS) [58] can be used to enhance the security mechanism in Open Grid Services Architecture – Data Access and Integration (OGSA-DAI) [2, 3]. In the OGSA-DAI, access control causes substantial administration overhead for resource providers in virtual organizations (VOs), because each of them have to manage a role-map file containing authorization information for individual Grid users. We used the 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.

In [61], we described how RBAC policies such as role privileges, constraints and hierarchies can be specified and managed for a community using CAS. Our access

control methods provide increased manageability for a large number of users and reduce day-to-day administration tasks of the resource providers, while they maintain the ultimate authority over their data resources. However, 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. Also, while our system in [60, 61] 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. It also does not provide support for the diverse policies across domains.

1.3. Contributions

The contributions of this dissertation are summarized as follows:

- We developed a role-based access control system for Data Grids using Shibboleth. By using Shibboleth, our system allows the user to control what personal information is released to the resource providers thereby supporting privacy protection.
- We used eXtensible Access Control Markup Language (XACML) and Object, Metadata and Artifacts Registry (OMAR) for the specification and administration of access control policies across diverse administrative domains. This allows for administering the access control policies uniformly across the diverse domains.
- We developed a semantic-based access control system using Shibboleth, XACML and ontology. We used OWL standard for specifying the semantics of the user attributes and data resources. By using semantics, the differences in the terminologies

used by the organizations can be resolved. Furthermore, the use of semantics enables the policies to be analyzed at different levels of abstraction.

- We developed an XACML policy administration tool for automatically generating XACML policies from existing RBAC policies. The tool also provides capabilities to update or create new RBAC policies.
- Our access control systems can provide increased manageability for a large number of users and reduce day-to-day administration tasks of the resource providers, while they maintain the ultimate authority over their resources.
- The implementation of the proposed systems with SRB and OGSA-DAI bring several advantages into its authorization infrastructure. The use of XACML policy specification dramatically reduces the number of entries to be managed in the authorization database and updates to them need to be made far less frequently.
- Our system evaluates each and every access request before establishing a connection to the resource. Thereby, allowing the resource to be available for authorized users.
- We also discuss the performance trade-offs of the proposed systems to the existing authorization mechanisms.

1.4. Outline of the Dissertation

The outline of the dissertation as follows: Chapter 2 describes the security in Data Grids. We also discuss the issues to be addressed while specifying access control in Data Grids. Chapter 3 reviews our completed work of the RBAC system in a Data Grid using

SRB, Shibboleth and XACML. Chapter 4 describes our completed work on semantic based access control system with OGSA-DAI, Shibboleth, XACML and ontology. Chapter 5 summarizes the finished works, and Chapter 6 contains the future work.

2. SECURITY IN DATA GRIDS

The three terms which are very important and often misunderstood in security are authentication, authorization and access control. Authentication is a process that verifies that the user is who he/she claims to be. Authorization identifies what permissions a user can have on which resource. Access control on the other hand refers to a more general way of controlling access to a resource, such as specifying restrictions based on time, location, etc.

These three terms can be explained in detail using the following example. Suppose you plan on traveling by an airplane. When you are at the airport you are asked to show your boarding pass along with a photo-identity, such as a driver's license or passport. The airport officer verifies the validity of the identity and verifies the name on the boarding pass with that on the identity. The officer also visually checks the picture on the identity card with your face to determine whether or not you are the person you are claiming to be. This process can be claimed as authentication. After successful authentication, you reach the gate at which you will be boarding a plane. The boarding pass you have contains the plane number you are supposed to board and the seat allotted to you.

Authorization is the process of allowing you to board a plane, which is specified in the boarding pass and take the seat you are allotted. If you try to board a plane other than the one mentioned in the boarding pass then you will not be authorized. Similarly, if you try to take a business class seat when you only purchased an economy you will not be authorized. Access control often goes together with authorization. Using access control the resources can control things such as the time at which you are allowed to access a resource. For example, an airport officer may only be allowed to work during certain timings. Several models such as mandatory access control, discretionary access control and role-based access control address the access control features such as mutual exclusion, time constraints, separation of duties, and so on.

This chapter contains a description of the security in Grids, technologies and authorization frameworks. As this research focuses on providing an access control mechanism in Data Grids, access control requirements in Grids are covered in detail. Outline of this chapter is as follows: Section 2.1 contains a description of the Globus Toolkit which provides the basic components necessary for establishing a Grid. It also describes the Grid Security Infrastructure (GSI), mutual authentication and single sign-on features. Section 2.2 contains a description of the access control requirements in Grids.

2.1. Globus Toolkit

Globus Toolkit [18] provides a set of basic services to establish a Grid system. The Globus Toolkit includes several widely used software components, shown in Figure 1, to provide security, data management, execution management, and information

services for establishing a Grid. These components use Web Services, which provide flexible, extensible and widely adopted XML-based mechanisms for interface [22]. The toolkit has recently been aligned with the Web Services Resource Framework (WSRF) [29]. WSRF defines conventions for managing the state in distributed systems based on Web services.

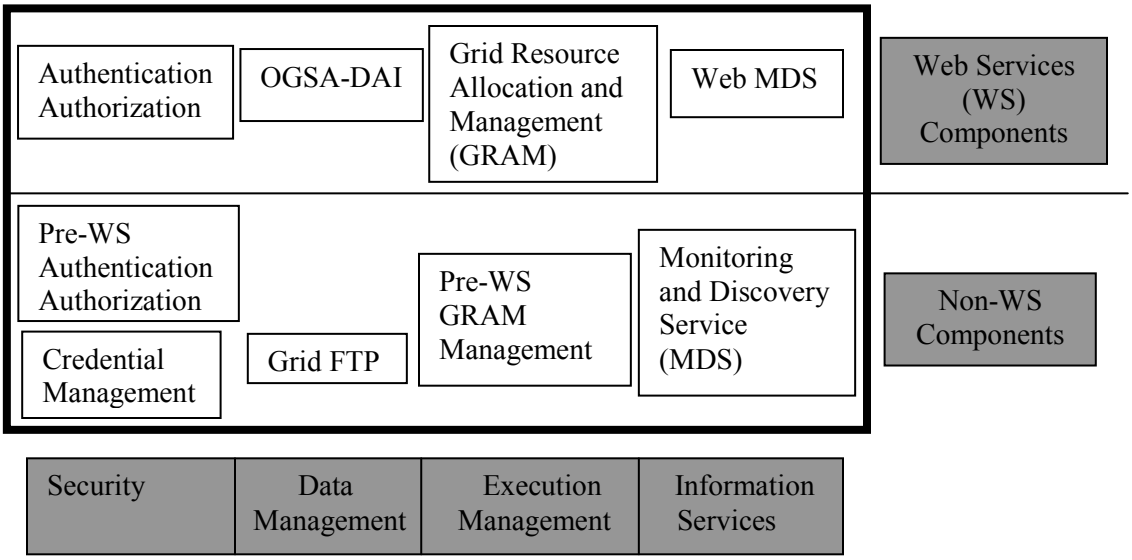

Figure 1: Basic components of Globus Toolkit version 4

The security components protect communications between users/resources and control the access to resources. They also provide message protection, delegation, authorization and authentication. The data management components provide solutions for reliable data transfer, data access and integration. The execution management components provide services for the resource allocation and management. The information services components support monitoring and discovery of resources through the use of Web Services Notification and WSRF. For each of these components, a C and/or Java Application Programming Interface (API) is available for developers [95].

2.1.1. Grid Security Infrastructure (GSI)

The Toolkit contains OpenSSL, which is used to create an encrypted tunnel between the clients and servers. The Toolkit offers the best-known security approach for Grids. It provides authentication, authorization and secure communication through the use of Grid Security Infrastructure (GSI). The major concepts of GSI are: a) to provide single sign-on for multiple Grid systems and applications, b) to provide secure communication between Grid systems and, c) to provide a security technology that can be implemented across different organizations without maintaining a central managing authority.

The GSI uses public key infrastructure (PKI) mechanisms which operate on credentials issued to every Grid user and service. PKI enables the users to establish confidentiality, message integrity and user authentication without having to exchange any secret information in advance. The credentials formed by an X.509 certificate and the associated public/private keys, are issued by a certificate authority (CA) trusted by all entities in a Grid. The Globus Simple CA package provides a convenient method of setting up a certificate authority that is interoperable with the Globus Toolkit. It provides the means for the generation of certificate requests and the signing of certificates [9]. It is capable of issuing certificates that can uniquely identify an individual user or an organization. The X.509 certificate consists of the 'subject' which is used to identify the certificate holder, the 'issuer' indicating the authority that has issued this certificate, the time during which this certificate is 'valid', in addition to the information pertaining to the public key of the user. The private key is held in a separate

file which is a read-only to the user. An example X.509 certificate issued by Globus simple CA is as shown in Figure 2.

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 3 (0x3)
      Signature Algorithm: md5WithRSAEncryption
      Issuer: O=Grid, OU=GlobusTest, OU=simpleCA-focus.cs.wright.edu,
       CN=Globus Simple CA
      Validity
         Not Before: Feb 16 19:38:13 2007 GMT
         Not After : Feb 16 19:38:13 2008 GMT
      Subject: O=Grid, OU=GlobusTest, OU=simpleCA-focus.cs.wright.edu,
       OU=cs.wright.edu, CN=Vineela Muppavarapu
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
         RSA Public Key: (1024 bit)
            Modulus (1024 bit):
               00:9c:50:05:da:19:9e:aa:7a:bc:54:43:29:4c:e0:
               56:59:8e:41:3e:a9:35:88:a7:b5:13:84:1a:4c:82:
               3b:33:14:2f:fb:48:4f:10:4d:e4:59:ec:ef:cc:c8:
               e0:cc:4d:7c:7c:2d:e4:15:48:a4:16:5e:48:bb:ef:
               1b:80:50:00:aa:e1:cc:cc:55:55:09:90:d2:b0:c5:
               fc:8d:7b:ca:52:0b:5d:19:85:c3:69:fb:82:2e:e0:
               7e:69:cd:09:99:a0:c5:60:54:5d:69:ff:e4:d6:3d:
               b7:fa:48:ff:97:d9:11:7b:05:64:c1:70:c0:d6:1a:
               a9:05:85:a0:32:18:75:74:93
            Exponent: 65537 (0x10001)
      X509v3 extensions:
         Netscape Cert Type:
            SSL Client, SSL Server, S/MIME, Object Signing
   Signature Algorithm: md5WithRSAEncryption
      3e:cb:ae:04:a4:ed:12:ec:bd:03:6f:18:fc:8e:f1:7d:d0:72:
      ea:49:10:88:d8:92:e4:54:9c:df:36:a3:f5:c6:43:83:3d:03:
      35:0b:a8:df:8d:c8:ea:e4:c9:67:c2:87:a9:e8:68:36:0b:59:
      17:26:59:9d:d7:62:0a:68:84:eb:cd:14:79:b8:be:7f:85:dd:
      ab:79:4f:d2:7b:98:c8:45:1f:13:01:c9:f5:5c:4f:03:60:54:
      24:61:75:35:3a:a6:45:8b:fc:d0:f2:a0:5f:57:ca:e9:0a:bd:
      f1:c8:2f:b7:a0:6a:d6:29:e3:2b:b5:64:1d:54:7f:c2:e0:95:
      9f:b2
```

Figure 2: X.509 certificate

2.1.2. Mutual Authentication and Single Sign-on

Grid Security Infrastructure (GSI) uses the Secure Sockets Layer (SSL) protocol for its certificate-based mutual authentication. Mutual Authentication is the process in which two parties can authenticate each other based on the credentials issued by some trusted third party, i.e., CA. Assume that there are two persons Alice and Bob are trying to establish mutual authentication then the process can be described as follows:

1. Alice establishes a connection to Bob.
2. To start the authentication process Alice sends her certificate to Bob. The certificate could be a standard X.509 certificate which contains the identity (subject) of Alice, her public key, and the CA that issued the certificate.
3. Bob will now check the validity of the certificate by verifying the CA's digital signature and the authenticity of the public key.
4. Bob then generates a random message and sends it to Alice.
5. Alice encrypts the random message sent by Bob using her private key and sends it back to Bob.
6. Bob decrypts the message using Alice's public key which was sent earlier in step 2. If the message is same as the message sent in step 4 then Bob knows that it is indeed Alice.
7. Now that Bob trusts Alice. Alice also checks for the validity of Bob's identity by repeating the steps 2-6 with person Bob sending his certificate and Alice sending the random message to Bob.

Figure 3 shows the use of proxy credential to support single sign-on in Data Grids. GSI offers users a secure authentication option by creating a proxy credential which is made up of a new public and private key pair and is signed by the user's private key. The proxy credential, issued by the end user not the CA, contains the user's 'subject' with a slight change to show that it is a proxy credential. The proxy credential is valid for a limited amount of time typically 12 hours. A user's proxy credential has to be created before contacting a Grid service. Once created, the proxy credential is used to grant or deny access to resources found throughout the Data Grid. Because the proxy is used across the system, this gives the end user the single sign-on capability.

The GSI handles user authorization by mapping the Grid user to a local user on the system being accessed. In a GSI-enabled Grid, the system receiving the request obtains the user's identity from the proxy, and then accesses a local file to map that name to a local user account. Thus, the Grid user does not need to have an individual account on the system he/she is requesting to access because the permissions for the Grid user would be same as those of the local user he/she is mapped to. By default, GSI provides secure communication using digital certificates for mutual authentication and SSL/TLS for data encryption.

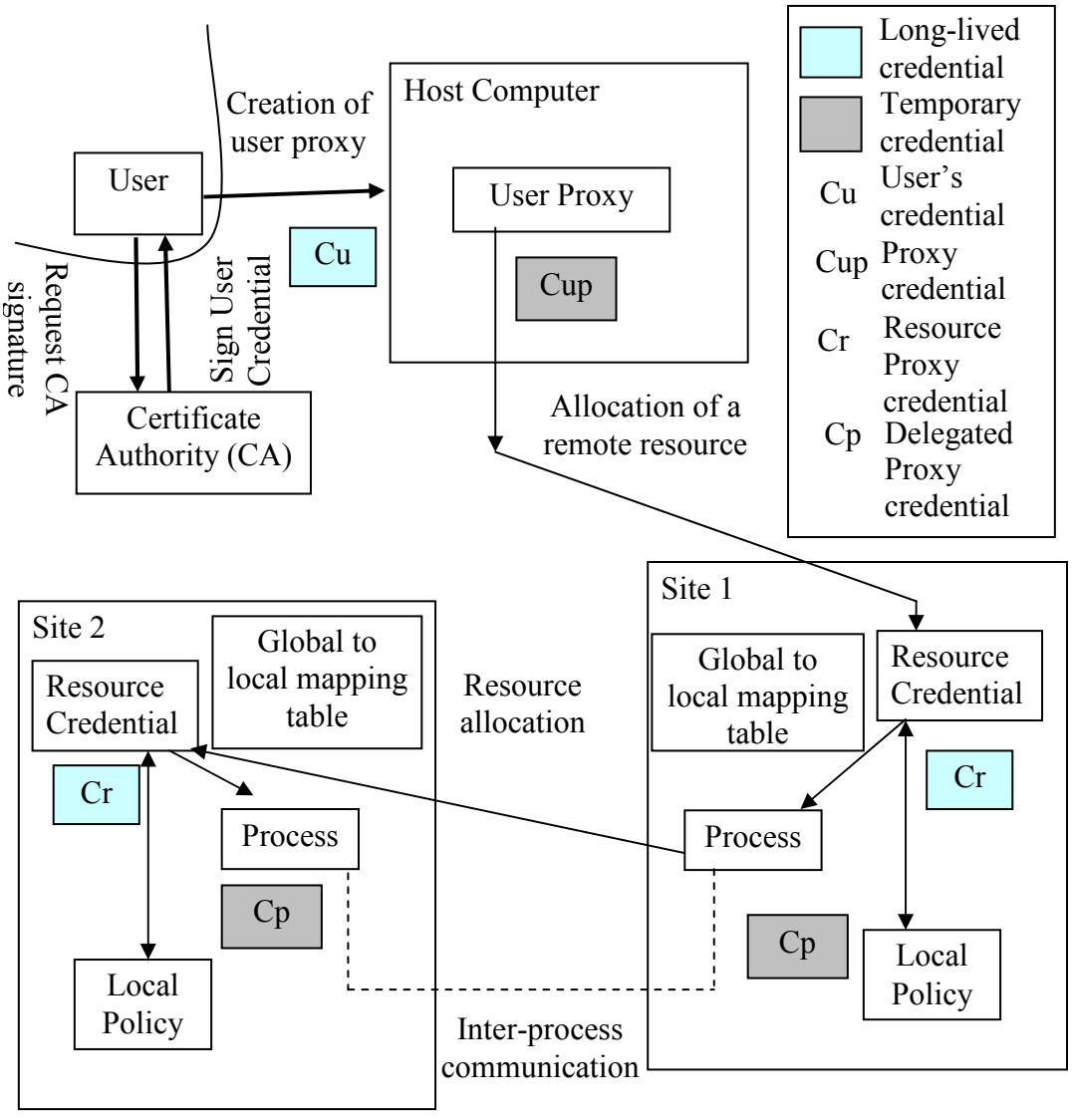

Figure 3: Use of proxy credential for single sign-on in Data Grids.

2.2. Issues to be Addressed While Specifying Access Control in Data Grids

Controlling the user access to the datasets and managing the users and data resources has become a crucial and prevalent issue for any distributed system. In Data Grids, as users and data resources are distributed across organizations securing the access to the data resources is becoming increasingly complex. In addition to authentication, integrity, single sign-on, and delegation capabilities, there are certain other requirements which have to be considered while specifying the access control for the Data Grid resources as outlined below:

Interoperability: In distributed environments, such as Data Grids, any data resource in the system can specify and enforce its own policy. There is no central policy and no central authority for allowing access to the data resources. Users and data resources may belong to multiple organizations with their own diverse security policies and mechanisms. Participating organizations may have different security models. It is important for these models to interoperate based on different levels of trust and contexts. Trust should be established not only among users and data resources, but also among the data 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 data resources are accessed or released [87]. Resource providers must understand and support the mechanisms and policies that are not strictly under their control.

Scalability: Based on the parameters used, scalability can be classified into two types: administrative scalability and performance scalability. For performance scalability, the

number of users is a major parameter. The system should not be a bottleneck and should scale even if the number of users increase significantly. For administrative scalability, the management of access control policies should scale even if users are added or deleted from the system. If the system is dynamic, administrative scalability is very critical [11]. In Data Grids, as users and data resources are dynamic, the permissions associated with the users and data resources have to be updated depending on the changes in the organization's policy. Also, as users join/leave the organization the administrator has to grant/refuse the necessary permissions. Considering the frequency of these changes and the number of users, the authorization mechanism should be scalable with regard to both performance and administration.

Privacy and Auditing: Privacy management and cross-domain auditing are significant challenges in distributed environments [28]. An audit mechanism is responsible for producing records which track security related events [45]. And, for this purpose, it is essential to keep a log of the user access requests and the enforced security policies. The user identity can be used to identify the user who initiated a request. The request can be logged at the data 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 [19]. However, this information could affect the privacy of users. For example, by examining the information logged at various sites with respect to the users belonging to a particular research group, it is possible to infer their data access patterns and thus obtain information about their work.

Principle of Least Privilege: Most systems do not enforce the Principle of Least Privilege [92]. A user or an application must be delegated only those privileges required for completing a certain set of tasks, otherwise the user/application should be totally trusted to do no more than required. In today's environment this is even more critical since software can be regularly downloaded from remote sites. In addition to the possibility of downloading malicious software such as Viruses, Trojans, Worms, and so on, we cannot expect software to work exactly as specified because of bugs or malicious intent. Any software with certain extra privileges has the potential to cause severe damage to computer systems and data.

3. ROLE-BASED ACCESS CONTROL IN A DATA GRID USING STORAGE RESOURCE BROKER AND SHIBBOLETH

In this chapter, we describe our role-based access control (RBAC) system for Data Grids using Storage Resource Broker and shibboleth. The Storage Resource Broker (SRB) [5, 64] was developed by the San Diego Supercomputer Center (SDSC) as a logical distributed file system based on client-server architecture. The SRB can be used as a data grid management system as it has features to support the management, collaboration, controlled sharing, publication, replication, transfer, attribute-based organization, data discovery, and preservation of distributed data.

The SRB stores and manages the access control information of individual users in the Metadata Catalog (MCAT) database. However, because of the specific MCAT schema structure, this information can only be used by the SRB applications. If an organization has many non-SRB applications, each with its own storage format for user

access control information, they cause substantial administration overhead because the system administrator has to manage them independently. Thus, multiple updates may be required if the access control information of a current user changes; and if new users are allowed to access many data resources, it can create a scalability problem with regard to administration. Additionally, as the access control information of each Grid user is maintained in the MCAT database, if users join/leave the organizations frequently, updating their access control information on an individual basis would be a difficult task. Moreover, a user may not want to reveal his/her personal information; hence, privacy protection becomes an important issue. The current identity-based access control mechanisms do not protect the privacy of the users. In our research, we developed a new access control mechanism which is not only scalable but also protecting the privacy of users.

Role-based access control (RBAC) [16, 65, 70, 72] is an approach where permissions are assigned to roles and roles are assigned to users, thereby acquiring the roles' permissions. Usually roles are created and assigned to users based on their job functions, and RBAC is more scalable than identity-based authorization because authorization information is associated with roles, not with individual users. In this paper, we propose a new RBAC system for heterogeneous data resources in the SRB by using Shibboleth [10] and the Object, Metadata and Artifacts Registry (OMAR) [53] as the main components.

Shibboleth is an attribute authorization service and is designed to provide the user attributes to the data resources for access control, and it mainly targets the internet-based resources. Shibboleth supports the authentication of user at his/her home institution. By

using Shibboleth, a user can either be authenticated at his/her home institution first and then present his/her authentication information and attributes to a Data Grid service, or contact a Data Grid service which is then redirected to the user's home institution to obtain the authentication information and attributes of the user. In either case, the user does not need to be authenticated at each individual site, and the individual sites do not need to maintain all the user information, as the authentication is handled by the user's home institution.

In Data Grids, users belonging to different organizations try to use the data resources and applications. If the SRB server can recognize the credentials issued by the user's home institution, then the SRB server does not need to individually manage the authentication information and attributes of each user. By using Shibboleth, the SRB server just needs to negotiate with the user's home institution regarding the security policies and then trust the authority that issues the credentials of the user. The authentication would then be handled by the user's home institution, and the required user information would be transferred to the SRB server. This approach would improve the interoperability among organizations, facilitate their collaboration, and relieve the SRB administrator from the burden of individually maintaining the authentication information of the users. Also, by using Shibboleth, access control policies can be specified based on the specific attributes of the user, such as role name and affiliation, instead of the identity of the user.

Shibboleth issues user attributes in the form of Security Assertion Markup Language (SAML) [47] assertion. In our access control system, this SAML assertion is embedded into the user's proxy credential. The SRB server can then obtain the SAML

assertion containing the role names of the user in addition to his/her identity or distinguished name (DN) from the proxy credential. If a system wishes to perform authentication using the individual identity or DN of the user, it can do so as the proxy credential contains the DN of the user. In our system, Shibboleth is also used as a Role Enablement Authority (REA), which is responsible for assigning roles to users and for activating roles within the user's session [48].

We used the Core and Hierarchical RBAC profile of the eXtensible Access Control Markup Language (XACML) [48] to specify the access control policies. XACML is a standard of the Organization for the Advancement of Structured Information Standards (OASIS) for describing the access control policies uniformly across different security domains [49]. Our system is based on the Core and Hierarchical components of the ANSI-RBAC [74], and the Core and Hierarchical RBAC profile of XACML defines how they can be specified in XACML.

For the distributed storage and administration of XACML policies and the user-role assignments, we used the OMAR. OMAR is an open-source implementation from the freebXML.org, and it provides an implementation of the OASIS e-business eXtensible Markup Language (ebXML) registry specifications [46]. A registry is an information system that securely manages any content type and the standardized metadata that describes the content. The ebXML registry specifications are developed to achieve interoperable registries and repositories with an interface that enables submission, query and retrieval [46].

Our system is scalable in terms of the number of access requests as well as the number of users; and it is robust as there is no single point of failure. It supports the

management of privileges and fine-grain attribute release policy; and it provides privacy protection for users. It can support a wide range of security policies using role-privileges, role hierarchies, and timing constraints. The MCAT database needs to maintain only the mapping information from Grid user roles to local roles and local policies, thus their administration overhead is reduced. When users join/leave, the MCAT administrator does not have to bother about individually adding/removing their information in the MCAT database, because OMAR can be used to directly grant/revoke the users' memberships on the roles.

We implemented our proposed RBAC system and analyzed its performance. A Shibboleth interface program (SIP) has been designed to obtain the user's attributes from the Shibboleth service and to create the user's Shibboleth proxy credential. The SRB client has been modified to send the user's Shibboleth proxy credential to the SRB server, instead of the proxy credential. The SRB server has been modified to obtain the user's attributes from the user's Shibboleth proxy credential and then to map the user's VO role to a local role by using the corresponding information in the MCAT database. Our performance analysis shows that the proposed system incurs a small overhead in setting up the security between the client and server. This overhead is quite acceptable when we consider the benefits of our system, such as scalability in managing access control policies and reduced administration overhead for the resource providers.

This chapter is organized as follows: Section 3.1 contains background information on SRB, RBAC and Shibboleth. It also describes the problems with the existing access control in SRB. Section 3.2 contains some related work on access control. Section 3.3 contains a description of our proposed RBAC system with Shibboleth and XACML for

SRB. Section 3.4 contains the implementation details and describes the changes we made on the client-side as well as the server-side. Section 3.5 contains the performance details with some graphs showing the effects of the changes made. Section 3.6 contains some conclusions.

3.1. Background

3.1.1. Storage Resource Broker (SRB)

The Storage Resource Broker (SRB) is developed as a client-server middleware that provides uniform interface to access heterogeneous distributed storage resources including file systems, database systems and archival storage systems [5]. The SRB presents the user with a single file hierarchy for the data distributed across multiple storage systems. The SRB provides the following: (1) a logical namespace to describe storage systems, digital file objects and collections; (2) specific features for digital libraries, persistent archive systems and collection management systems; (3) capabilities for storing replicas of data, authenticating users, controlling access to documents and collections, and auditing accesses; (4) user-defined metadata management at the collection level and object level, and search capabilities based on the metadata [63].

SRB is a comprehensive distributed data management solution and has features to support the management and collaborative (and controlled) sharing, publication, replication, transfer, and preservation of distributed data collections. The SRB also serves as a middleware via a rich set of APIs available to higher-level applications.

SRB, in conjunction with the Metadata Catalog (MCAT) database, supports location transparency by providing access to data resources based on their attributes rather than their names or physical locations. The MCAT database contains file attributes, metadata, user information, security and access control information, and also maintains audit trail on data and collections of data. The architecture, entities and the attributes of MCAT database are described in [41].

SRB enables easy access across a variety of storage devices and provides improved discovery, access and management of data files. Users and administrators benefit from an enterprise-wide, transparent view of all data. SRB uses the Metadata Catalog (MCAT) and provides access to data resources based on their attributes rather than their names or physical locations. The MCAT server uses a database to store the system and dataset metadata. Metadata includes the physical and logical details of the data held and its replicas, user information, security and access control information.

The MCAT is used to determine where a given data object is physically located, and which drivers handle and access the data of interest. Furthermore, the MCAT contains all file attributes, metadata, access control lists (ACLs), storage resource information, and user data, and also handles queries.

3.1.1.1. The SRB Architecture

The SRB system is a distributed system comprised of three major components, MCAT database, SRB server and SRB client. The architecture of the SRB system is as shown in Figure 4. Usually the SRB system consists of a single MCAT database and one

or more SRB Servers and Clients. If the SRB server is directly linked to the MCAT database then it is referred as MCAT-enabled SRB server (MES server) otherwise it is called a non-MCAT-enabled SRB server (non-MES server). The advantage of MCAT architecture is that users can transparently access data no matter where the data objects are located or what hardware or software they are stored on.

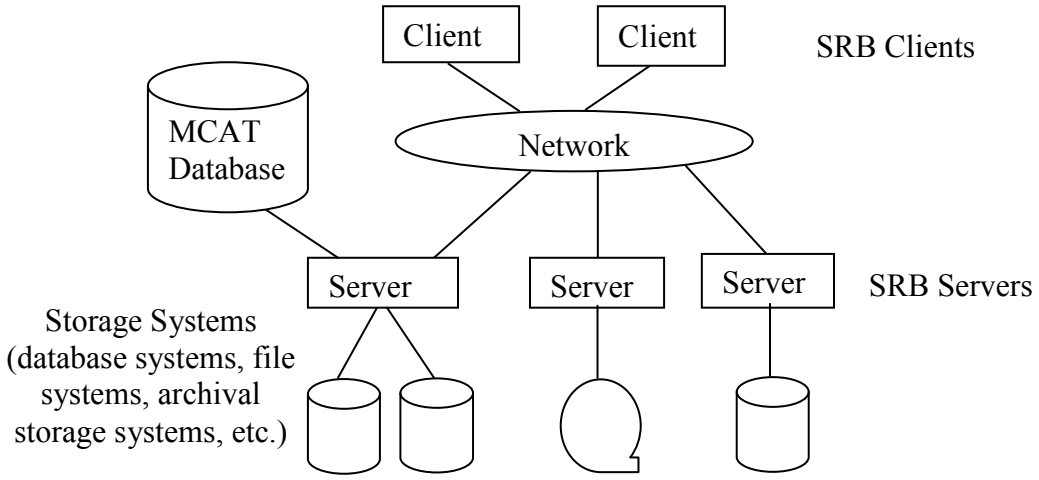

Figure 4: SRB system architecture overview

Two or more independent SRB systems can interact with each other and allow for access to the data and metadata. These systems are called zones or federated MCAT system. Each of the SRB system maintains its own MCAT to store all the metadata information, corresponding to that zone. However, in case of federated MCAT system, this information has to be updated frequently across the other zones' MCAT enabled Servers (MESs) to avoid any inconsistencies in the metadata and access control information stored in the MCAT. Thus, all the MCATs in the federation would be maintaining the same information and even if the local MCAT fails the information can be accessed from the other zones' MCAT database.

SRB provides command line utilities, called *Scommands*, for interacting with the SRB server. Scommands are the Unix-like commands in which most of the command names are preceded by S. These are Unix-like commands for accessing SRB data and metadata. For example, the command *Sls* works similar to the way *ls* command works on a Unix machine.

3.1.1.2. Security and Access Control in SRB

SRB supports two types of authentication: the ENCRYPT1, which is a password-based authentication system, and the Grid Security Infrastructure (GSI). The user passwords for ENCRYPT1 and the distinguished names (DNs) for GSI are stored in the MCAT database. The following is a brief description of how the security in SRB system works. The client application contacts the SRB server and requests the data or a set of tasks to be performed on the data. The server then contacts the MCAT enabled server (MES) to see if the request is from an authenticated user. The MES checks the MCAT database and authenticates the user based on the user's identity. Once the user is authenticated, the user is allowed to see all the metadata he/she is authorized to see. The user can only see the metadata describing the data objects which he/she can access. The tasks requested by the user would then be performed if the user has the permission to access the data sets.

SRB supports two types of access control: a) Unix-type access permissions that allow read, write and all access permissions to be set on a dataset or collections of data. It is also possible to allow/deny access to as many groups and users as needed. b) A

ticketing system, in which, tickets are issued by owners of dataset to other users, groups of users and everyone. Owners of the data objects can specify which users and user groups can have what permissions (read, write, etc.) on their data objects. For all the data objects exposed by an SRB server, permissions assigned to an individual user/user group should be stored in the MCAT database. If the privileges of a user change in an organization, the corresponding information should be updated in the MCAT database.

The ticket mechanism, supported by SRB, is a restricted form of read privilege [6]. The user with the privilege of granting a ticket on a data object can create tickets and issue them to other users. This ticket could be valid for a specific period of time or for a certain number of uses, thus the read access to a data object can be controlled. This ticket mechanism can only control read accesses, thus it limits the delegation of access privileges. Furthermore, the current access control system of the SRB is not scalable in a Data Grid environment, especially if the resource providers and users belong to multiple organizations, because it would be very difficult to manage the access control information for individual users as well as their tickets. Most of all, the access control information stored in the MCAT database cannot be used by non-SRB applications.

3.1.2. Role-Based Access Control

In role-based access control model, roles are associated with permissions and users are assigned roles thereby users acquire the roles' permissions. Since RBAC was proposed it has been used extensively in scientific as well as business communities. Supporting role-based access control (RBAC) [16, 70] is desirable in Data Grids. RBAC

shows clear advantages over traditional discretionary and mandatory access control models in such environments, because it allows a uniform representation of diverse security policies [33]. However, additional actions should be taken to make sure that inter-domain violations do not occur.

In RBAC, permissions are associated with roles, and users are assigned appropriate roles, thereby acquiring the roles' permissions [65]. 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 privileges a user can have; for example, access to a specific table of a database.

In VOs, users may be assigned specific tasks, and there may be constraints related to the execution of those tasks. For example, a user may have access to data only during certain days of the week, or certain tasks may be considered mutually exclusive for a user; i.e., any two or more tasks cannot be executed at the same time. RBAC can support a wide range of security policies using role-privileges, role hierarchies, and constraints.

The typical identity-based authorization used today is not scalable because authorization information should be maintained for each user. In RBAC, authorization information is associated with roles, not with individual users. It has been shown that the cost of administering RBAC is proportional to $U+P$ per role, where U is the number of individuals in a role and P is the number of permissions required by the role, while the cost of associating users directly with permissions is proportional to $U \times P$ [17, 93].

In certain instances, a user may wish to delegate only a subset of its rights to an application to act on its behalf. This requirement can usually arise in systems where a

limited trust relationship is established between entities. For example, a user may contact a data mining service to mine certain data sets that the user has access to. If the trust between the user and the service is limited, then the user may want to delegate only a specific subset of its rights to the service, thus enabling it to complete only the required task and nothing more. With RBAC, such delegation could be done easily. For example, a user in a special role can delegate privileges to other roles.

RBAC is distinguished by its inherent support for the Principle of Least Privilege [40], which requires that a user be given no more privileges than necessary to perform a job [16]. 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. Users can request a particular role among those they are entitled to and, hence, gain the specific permissions tied with that role. Furthermore, current RBAC models are modular and can thus incorporate sophisticated functionality such as RBAC policy administration. Also, more complex forms of access control, such as task-based access control (TBAC), can be layered on RBAC [70].

3.1.3. Shibboleth

Shibboleth [10] is an attribute authorization service, developed by the Internet2 community for cross-organization identity federation [88]. Shibboleth creates a distributed authorization infrastructure for web resources, and simplifies access control policies and their management [4]. It enables anonymous interaction between users, thus protecting individual privacy while still providing basic security for resource providers

[88]. 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 [28].

A target Data Grid service authenticates a user by using GSI, determines the address of the appropriate Shibboleth attribute service in the process, and then obtains the selected user attributes, that the Data Grid service is authorized to see, from the Shibboleth service. These attributes are presented to the requested service by using Security Assertion Markup Language (SAML) [47] attribute statements. The attributes thus obtained will then be used by the Data Grid service in making authorization decisions. These attributes will be passed securely through a trust relationship to the Data Grid service. To provide anonymity in the Data Grid context, users are issued a set of credentials with pseudonym identifiers, and they will have the option of releasing only a subset of their attributes to particular resources. For example, identifying information about the user does not need to be released.

3.1.4. eXtensible Access Control Markup Language (XACML)

XACML is an OASIS standard for describing access control policies uniformly across different security domains [49]. XACML defines an access control framework as shown in Figure 5. The framework consists of a Policy Administration Point (PAP), Policy Information Point (PIP), a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP). The basic functionality of these components is as follows: A PIP releases

attribute values related to the subject (such as a user, application), the resource and the environment. A PDP makes an authorization decision based on the attributes released by the PIP and the policy maintained by the PAP. PEP executes the decision of the PDP by either performing or denying the client's request.

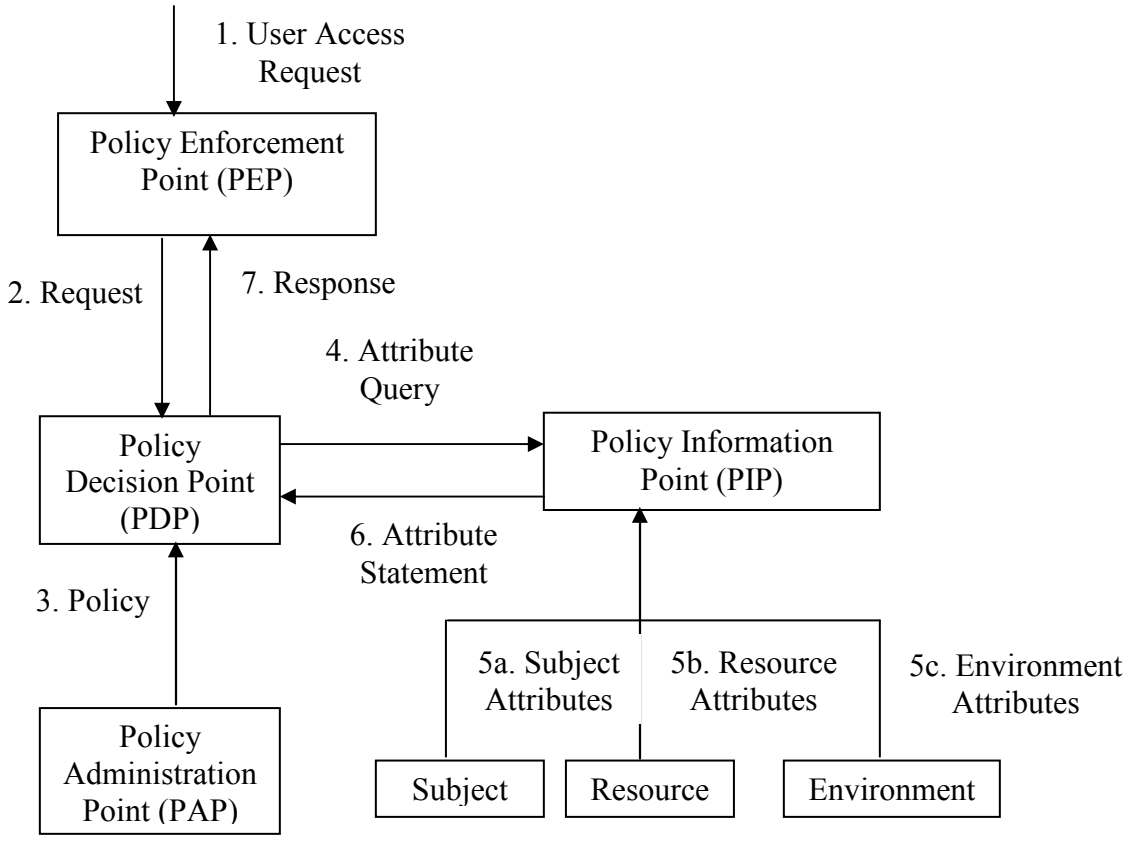

Figure 5: XACML authorization model

The data flow in the XACML authorization model, shown in Figure 5, can be described as follows: A PEP receives the access requests from the user and forwards them to the PDP. The PDP retrieves the policy from the PAP and sends an attribute query to the PIP requesting the attributes of the user. The PIP then obtains the user attributes from the policy database or file and sends the attributes back to the PDP in the form of an

attribute statement. The PDP then compares the policy retrieved from the PAP with the attributes released by the PIP and makes a decision either “permit” or “deny” and sends it to the PEP. The PEP receives the decision from the PDP and enforces the decision either by granting the access request of the user or by denying the request.

XACML also defines a policy language with the following main components to represent the policies:

(1) A <PolicySet> contains a set of policies or other policy sets.

(2) A <Policy> represents an access control policy described through a set of rules.

(3) A <Rule> represents an access rule or permission.

An XACML <PolicySet>, <Policy> or <Rule> may contain a <Target> element. A <Target> element specifies the set of subjects, resources, actions and environments to which the <PolicySet>, <Policy> or <Rule> applies [49]. The Core and Hierarchical RBAC profile of XACML specification defines how core and hierarchical RBAC defined in [74] can be specified in XACML. The Core and hierarchical profile further defines the following components:

(1) Permission <PolicySet> or PPS contains <Policy> elements and <Rules> associated with a given role. A Permission <PolicySet> may also contain references to other Permission <PolicySet>s associated with other roles that are junior to the given role, thereby allowing the role to inherit all the permissions associated with the junior roles. The <Policy> elements and <Rules> of the PPS

describe the resources and the permissions on the resources, along with any conditions on those permissions.

(2) Role <PolicySet> or RPS associates a role with the corresponding PPS. Each RPS can only refer to a single PPS.

(3) Role Assignment <Policy> or <PolicySet> defines which roles can be enabled or assigned to which subjects.

3.2. Related Work

We use the Shibboleth Identity Provider (IdP) to support RBAC in SRB, as described later in Section 3.4. Shibboleth has certain advantages, such as privacy protection, over other authorization services for Grids, such as the Community Authorization Service (CAS) [58], Virtual Organization Management Service (VOMS) [1], Akenti [82] and PERMIS [54].

CAS provides a scalable mechanism for specifying and enforcing complex and dynamic policies that govern the access to Grid resources. CAS allows resource providers to delegate some of the authority for maintaining fine-grain access control policies to communities. CAS issues authorization assertions containing the list of actions the user is allowed to perform on a resource. However, CAS being a centralized authorization system proves to be a bottleneck.

VOMS supports user-role assignments and provides attribute certificates, containing user's VO roles and group memberships, to the users. The attribute certificates do not provide rights directly, and they need to be interpreted by the resource. 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 [82]. 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 [20].

PERMIS [54] is an attribute-based authorization service, and so is VOMS. They use assertions that bind the attributes to users for authorization, as opposed to the typical identity-based authorization used today [24]. However, currently they do not support any standard for how attributes are transferred from the attribute authority to the Grid services and no standard is used for expressing the policy regarding those attributes [24].

3.3. An RBAC System with Shibboleth for the SRB

In this section, we propose a new RBAC system using Shibboleth for the SRB system. The Core and Hierarchical RBAC profile of the eXtensible Access Control Markup Language (XACML) [48] is used to specify the access control policies. For the distributed storage and administration of XACML policies, including user-role assignments, we used the Object, Metadata and Artifacts Registry (OMAR).

3.3.1. Drawbacks of the Existing Access Control Method in the SRB

The SRB server authenticates the user based on his/her individual identity. Upon successful authentication, the SRB server checks whether that identity is authorized to

perform the requested action, such as accessing a file, creating a new file/directory, etc. However, this method is not suitable when the number of users is large in a SRB system, because the MCAT database server could become a bottleneck. Moreover, when a local security policy is changed, it may require the MCAT database to be updated.

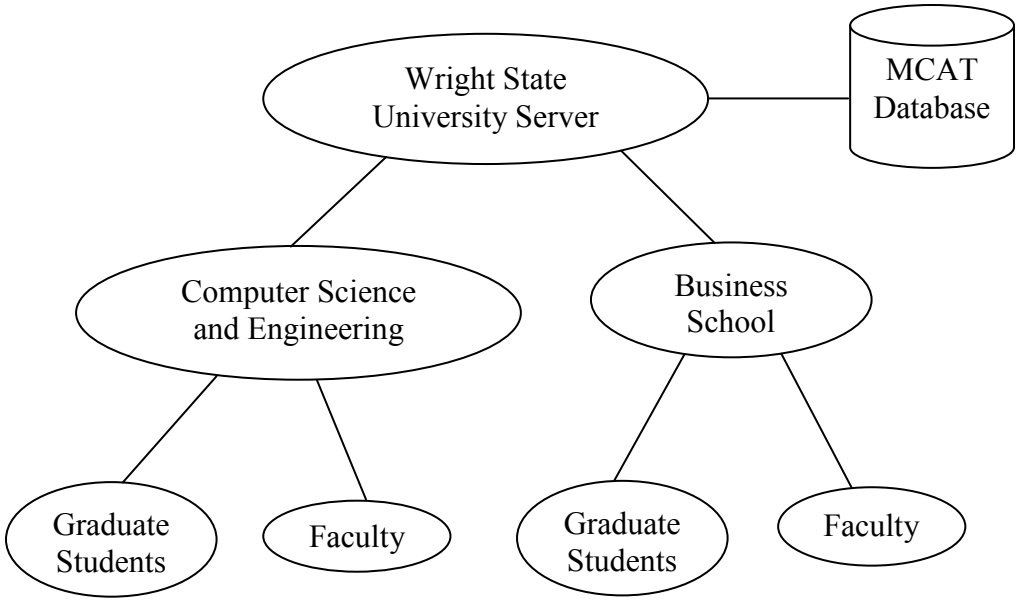

Figure 6: Scenario describing the problem with current implementation

To illustrate this access control problem in a SRB system, let us consider the following example: Suppose that a university created a Data Grid by installing an MCAT database and an SRB server at the university level. For simplicity, assume that the Wright State University establishes a Data Grid by installing an MCAT database and an MCAT enabled server at the university level. For simplicity, let us further assume that it has two sets of data, one of which can be accessed by the graduate students and faculty of the Computer Science and Engineering (CSE) department and the other dataset can be

accessed by the graduate students and faculty from the business school. These datasets may also be accessed by students and faculty from other departments.

With the current implementation, the MCAT database administrator will have to manually enter all the identities (for example, the 'subject') of the graduate students and faculty from the CSE department as well as those from the business school to allow the access to the corresponding data resources. If any student/faculty joins/leaves/changes the department his/her identity ('subject') has to be added/deleted/updated in the MCAT database. Considering the frequency of those personnel changes the MCAT database server has to be updated quite often, and it increases the burden on the administrator. If the university decides to collaborate with other universities to form a larger Data Grid, the administration overhead would increase multiple folds.

For improving the scalability of the SRB system a federated system or Zone SRB has been proposed. In a federated system each server maintains its own MCAT database. A zone can have a single MCAT database. All the zones (MCAT databases) may be federated and the data may be replicated depending on the requirements. Certain zones may wish to share its data while certain zones might share the access control information of the users. This solution reduces the latency of the requests and also seeks to address the bottleneck problem. However, an additional database has to be administered and managed which might not be an ideal solution in many cases. Also, the access control information stored in this database may not be usable to other applications. There is no support for leveraging the existing security infrastructure.

3.3.2. Our Proposed RBAC with Shibboleth

Our proposed system uses the VO roles of the Grid users instead of their identities. By using the VO roles, the university server only needs to maintain few roles instead of all the user identities. This also reduces the administration overhead for the MCAT administrator as he/she does not have to worry about individually adding or removing the user identities from the database.

In our system, Shibboleth maintains the security policies of the VOs, grants the users' memberships on VO roles, and then authorizes them to use the privileges of those roles. The data resource providers only need to maintain the mapping information from the VO roles to the local roles and local policies, thereby reducing the number of entries to be maintained in the MCAT database dramatically. Our system also allows the specification of policies at the VO level, thus if the users do not possess the required privileges, their access can be denied at the VO level. This eliminates unnecessary authentication, mapping and connection overheads on the resource providers. When users join/leave the VO, Shibboleth can just grant/revoke their memberships on the VO roles.

Our system architecture is shown in Figure 7. Shibboleth releases the attribute values related to the user. The SRB server obtains the user attributes released by Shibboleth, and then makes an authorization decision based on the user attributes and the corresponding resource policy information. Based on its decision, the SRB server either performs or denies the client's request.

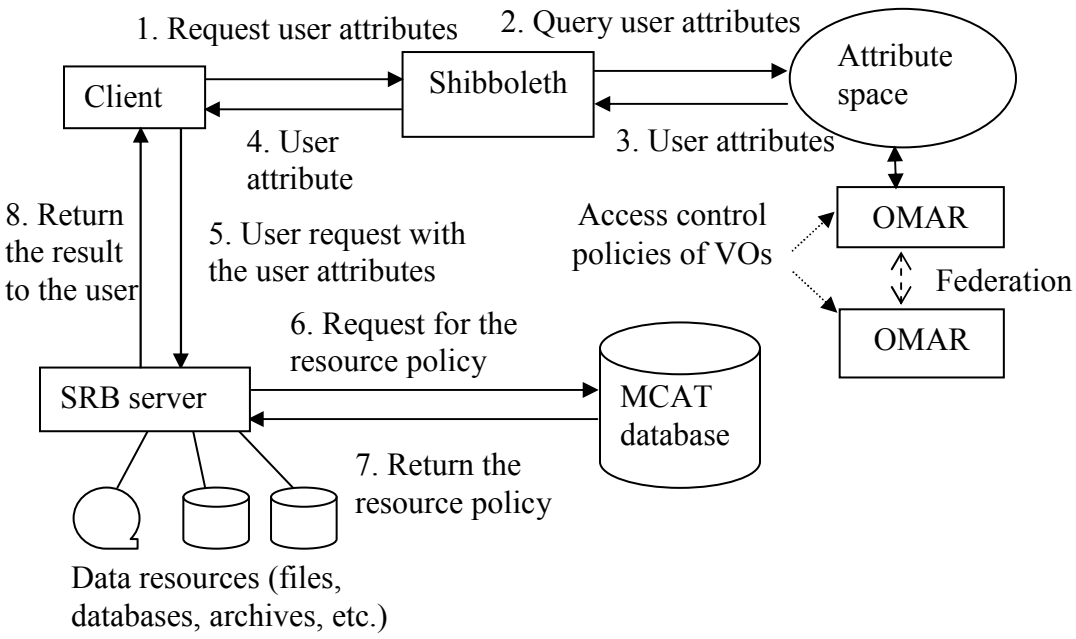

Figure 7: Proposed architecture

The data flow in our system is described in detail as follows:

1. The client sends an attribute query (as a SAML request) to Shibboleth.
2. To obtain the user information, Shibboleth queries the OMAR registry, which stores the user-role assignments and the access control policies in XACML.
3. The user attributes are returned from the OMAR registry.
4. Shibboleth sends the SAML response containing the user attributes to the client.
5. The client makes an access request to the SRB server along with the user attributes.

6. The SRB server obtains the user attributes from the client, and then queries the MCAT database to obtain the corresponding resource policy information, including the mapping of the VO role to local roles and their access permissions.
7. The MCAT database returns the resource policy information to the SRB server.
8. If the resource policy allows the access, then the client request is performed by the SRB server, otherwise it is denied.

Our system works in the push model, as the user can directly obtain the permissions from the authorization service and then pass them to the SRB server at the time of making a request. The SRB server then verifies the authenticity of the user and authorizes the user based on the permissions obtained, provided the authority that issued them is trustworthy. An advantage of the push model is that it allows the user to explicitly select a role from the set of roles he/she is entitled to. Additionally, the push model is inherently scalable because the SRB server does not need to contact the authorization service to obtain the user attributes for each access request. In case that the user and the authorization service belong to the same organization and are protected by a firewall, the push model should be deployed because the SRB server may not be able to contact the authorization service directly.

Shibboleth also supports the pull model, where the user directly contacts the SRB server first. The SRB server then contacts Shibboleth 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 [88]. However, considering the advantages of the push model, our system is currently deployed in the push model alone.

In traditional systems, an organization's entire security policy is managed by a single central authority; however, in a VO, its security policy is distributed across its member organizations. Thus, it may be necessary to manage the user attributes across different organizations and to aggregate them for an authorization decision. Shibboleth allows the VO to manage different subsets of its attribute space at different member organizations, as each member organization operates a Shibboleth service. In addition, the local access control policies of each organization can be managed by OMAR, which also allows the federation of the local access control policies of different organizations. Thus, distributed administration of the access control policies of a VO is feasible, while maintaining the local autonomy of each member organization. This also ensures the privacy of users, as the individual user information is solely administered at the user's home organization.

3.3.3. Specification of RBAC Policies Using XACML

The specification of RBAC policies for the VO roles can be done by the VO administrator using the XACML policy language. A single policy can include any number of decentralized rules, each managed by different organizations [38]. In multi-domain environments such as VOs, it is necessary to distinguish the user attributes in a VO from the user attributes in other VOs. For example, the employee role in the Accounting subgroup of the VO Alpha may have different permissions from the employee role in the Accounting subgroup of another VO Beta. Hence, when making an authorization decision, it is important to know not only the role name employee but also the VO name, either Alpha or Beta.

In order to manage and identify the attributes from different domains, Shibboleth uses scoped attributes, defined in SAML, which can include the domain name. A scoped attribute is a combination of a value and its scope. The scope identifies the domains and sub-domains in which the values are defined. An example scoped attribute is "faculty@abcuniv.edu", which identifies the value "faculty" in the scope "abcuniv.edu". However, the XACML profile does not support the scoped attribute values for subjects such as roles. Mapping SAML to XACML allows the systems using XACML to store SAML attributes [51].

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicySetId="RPS:employee:role" PolicyCombiningAlgId="&policy-
combine;permit-overrides">
   <Target>
      <Subjects>
        <Subject>
        <SubjectMatch  MatchId="&function;anyURI-equal">
            <AttributeValue DataType="&xml;anyURI"
>&roles;employee</AttributeValue>
            <SubjectAttributeDesignator AttributeId="&role;"
DataType="&xml;anyURI"/>
        </SubjectMatch>
        </Subject>
      </Subjects>
   </Target>
   <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
</PolicySet>
```

Figure 8: RPS of employee role

The RPS of the employee role, shown in Figure 8, references the PPS of the employee role via <PolicySetIdReference>. The PPS of the employee role is shown in Figure 9. The RPS of the manager role is not shown here, but is similar to the RPS of employee, except that the role name is manager and the <PolicySetIdReference> references PPS:manager:role shown in Figure 10.

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicySetId="PPS:employee:role"
PolicyCombiningAlgId="&policy-combine;permit-overrides">
  <Policy PolicyId="Permissions:specifically:for:the:employee:role"
RuleCombiningAlgId="&rule-combine;permit-overrides">
   <Rule RuleId="Permission:to:read:data:from:DataSet1" Effect="Permit">
    <Target>
     <Resources> <Resource>
      <ResourceMatch MatchId="&function;string-equal">
        <AttributeValue DataType="&xml;string">DataSet1</AttributeValue>
        <ResourceAttributeDesignator AttributeId="&resource;resource-id"
DataType="&xml;string"/>
      </ResourceMatch>
     </Resource> </Resources>
    <Actions> <Action>
         <ActionMatch MatchId="&function;string-equal">
        <AttributeValue DataType="&xml;string">read</AttributeValue>
            <ActionAttributeDesignator AttributeId="&action;action-id"
DataType="&xml;string"/>
          </ActionMatch>
      </Action> </Actions>
   </Target>
   <Condition>
     <Apply FunctionId="&function;and">
         <Apply FunctionId="&function;time-greater-than-or-equal">
        <Apply FunctionId="&function;time-one-and-only">
            <EnvironmentAttributeDesignator
AttributeId="&environment;current-time" DataType="&xml;time"/>
        </Apply>
        <AttributeValue DataType="&xml;time">9h</AttributeValue>
          </Apply>
         <Apply FunctionId="&function;time-less-than-or-equal">
           <Apply FunctionId="&function;time-one-and-only">
           <EnvironmentAttributeDesignator AttributeId="&environment;current-
        time" DataType="&xml;time"/>
           </Apply>
           <AttributeValue DataType="&xml;time">17h</AttributeValue>
         </Apply>
        </Apply>
    </Condition>
  </Rule>
 </Policy>
</PolicySet>
```

Figure 9: PPS of employee role

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicySetId="PPS:manager:role"
PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Policy PolicyId="Permissions:specifically:for:the:manager:role"
RuleCombiningAlgId="&rule-combine;permit-overrides">
  <Rule RuleId="Permission:to:write:data:to:DataSet1" Effect="Permit">
    <Target>
     <Resources>
      <Resource>
       <ResourceMatch MatchId="&function;string-equal">
        <AttributeValue
DataType="&xml;string">DataSet1</AttributeValue>
         <ResourceAttributeDesignator AttributeId="&resource;resource-id"
DataType="&xml;string"/>
        </ResourceMatch>
       </Resource>
    </Resources>
   <Actions>
      <Action>
       <ActionMatch MatchId="&function;string-equal">
            <AttributeValue
       DataType="&xml;string">write</AttributeValue>
            <ActionAttributeDesignator AttributeId="&action;action-id"
       DataType="&xml;string"/>
       </ActionMatch>
      </Action>
     <Actions>
    </Target>
   </Rule>
  </Policy>
  <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
</PolicySet>
```

Figure 10: PPS of manager role

A policy could contain the individual permissions and timing constraint information of a role, indicating that the user in that role can access a resource during a specific time period. In [34], the authors introduced a specification of the periodicity and duration constraints in RBAC. Our system can use this specification, and the timing constraints can be represented in the XACML policy by using the Environment attributes.

For example, Figure 9 contains the periodicity of the time duration between 9:00 AM and 5:00 PM each day. The PPS of the employee role grants the permission to execute the read operation (specified within <Action>) on the resource 'DataSet1' (specified within <Resource>) only from 9:00 AM to 5:00 PM (specified within <Condition>). The PPS of the manager role, shown in Figure 10, grants the permission to execute the write operation on 'DataSet1'. It references the PPS of the employee role via <PolicySetIdReference>, thereby inherits all the permissions of the employee role.

3.3.4. Object, Metadata and Artifacts Registry (OMAR)

For the storage and management of the XACML policies and the user-role assignments, we used the Object, Metadata and Artifacts Registry (OMAR) [53] which provides an implementation of the OASIS ebXML registry specifications. The ebXML specifications are developed to achieve interoperable registries and repositories, with an interface that enables submission, query and retrieval of the registry and repository contents [46]. An ebXML registry is an information system that securely manages any content type and the standardized metadata that describes the content. It also provides a set of services for sharing of its content and metadata between organizations in a federated environment [50].

OMAR stores data in a repository and stores the associated metadata as registry objects [50]. The relationship between registry objects is represented by an association object. OMAR allows many-to-many associations between registry objects. OMAR uses an object, called slot, to add attributes dynamically to registry objects.

3.3.5. Administration of the RBAC Policies in XACML by Using OMAR

The XACML role policy information of individual users can be managed by the OMAR. OMAR stores the XACML policies in their entirety as repository items in a relational database and classifies them as either XACML Policy object or XACML PolicySet object. However, as policies are stored in their entirety, any updates in policies become difficult especially for VOs whose policies are complex and tend to change dynamically. To solve this problem, we propose to split each policy into components and store them as different objects, as shown in Figure 11.

OMAR supports the federation of registries by defining a federation object and by creating an association between the registry objects and the federation object. Federation allows multiple registries to link together seamlessly and appear as a single logical registry, while retaining local autonomy and security. Thus, XACML-based policies can be stored and managed across multiple sites in the Data Grid, while each site maintains its own registry and Shibboleth service. Hence, there is less chance to have a bottleneck, and there is no single point of failure in the system as attribute queries are distributed among the Shibboleth services.

One of the key features of RBAC is the ability to manage itself through administrative roles and permissions [71]. Users in administrative roles can create roles and role hierarchies; make user-role and permission-role assignments; and specify constraints. Furthermore, they can grant administration privileges to other users. This administration feature of RBAC is not directly addressed by the RBAC profile of XACML, but can be easily realized through OMAR.

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" PolicySetId="PPS:employee:role"
PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Policy PolicyId="Permissions:specifically:for:the:employee:role" RuleCombiningAlgId="&rule-
combine;permit-overrides">
  <Rule RuleId="Permission:to:perform:read:on:DataSet1" Effect="Permit">
    <Target>
      <Resources>….DataSet1… </Resources>
      <Actions>…read… </Actions>
    </Target>
  </Rule>
 </Policy>
</PolicySet>
```

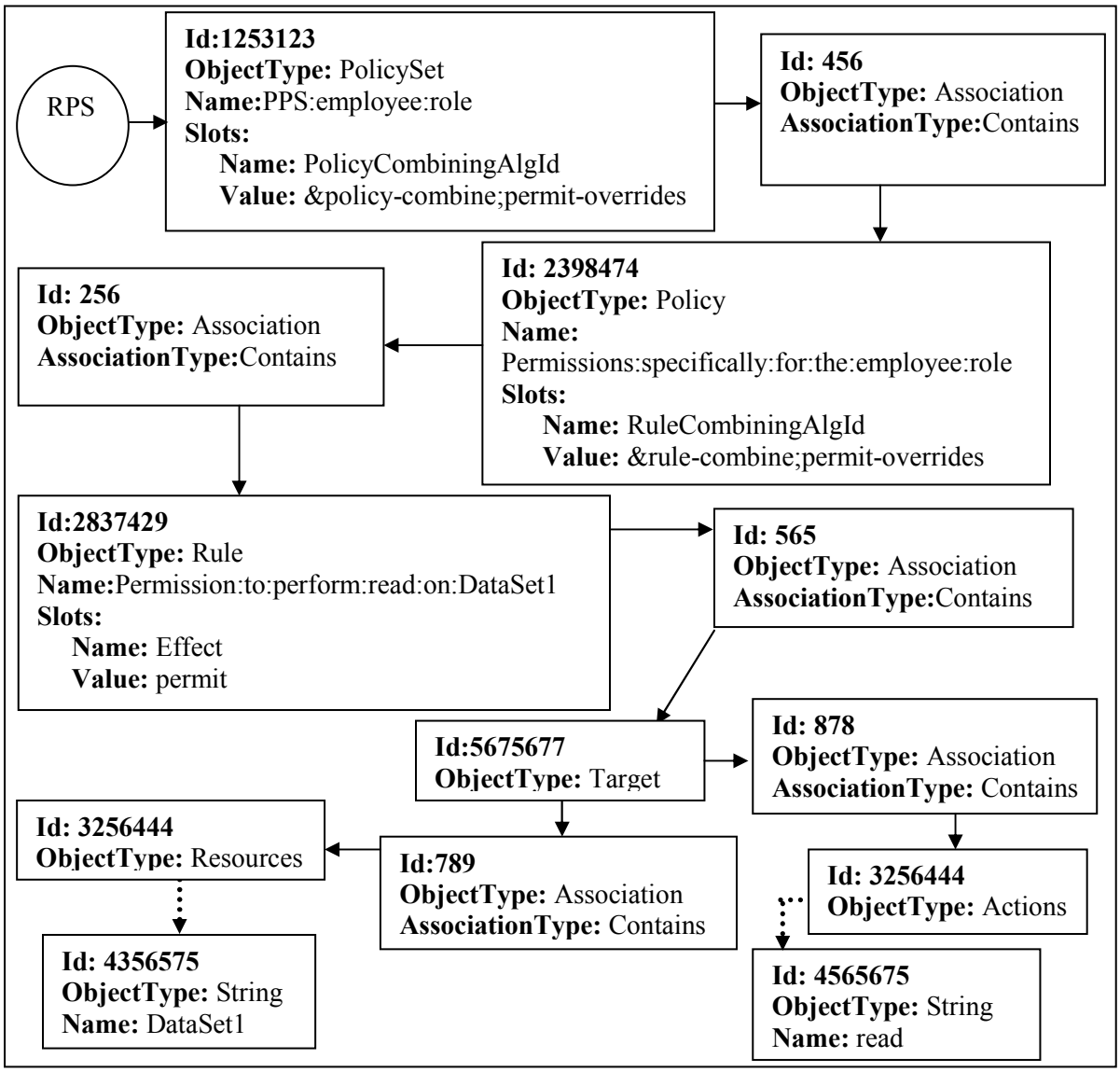

Figure 11: A part of employee PPS and corresponding storage in OMAR

A user with OMAR registry administration privileges can create policy objects and determine how other registry users can access them through the Access Control Policy (ACP) file. In particular, that user can create a registry role (by creating policy objects) and assign the privilege of creating/accessing certain registry objects to that role. A registry role is an OMAR component similar to a role related to a database. The user who creates a registry role can also grant memberships on that registry role to other registry users. For example, a VO administrator having OMAR registry administration privileges can create a new registry role and assign (to that role) only the privilege of creating users in VO employee role. The VO administrator can then assign a VO manager to the new registry role (after creating a registry account for the VO manager). Then, the VO manager can create new users in VO employee role by himself/herself.

3.4. Implementation Details

Our RBAC system with Shibboleth supports the push model, where the user directly obtains his/her authorization information, in terms of VO roles, from Shibboleth and passes them to the SRB server at the time of making a request. We have designed a new program, called *Shibboleth interface program* (SIP), which can be invoked by the user to obtain the authorization information from Shibboleth. This program creates a Shibboleth proxy credential by including the authorization information into the user proxy credential. The user proxy credential is formed by an X.509 certificate and the associated public/private keys. The X.509 certificate is issued by a certificate authority (CA) trusted by all entities in the Grid [8]. As the user proxy credential is valid for a

limited period of time, 12 hours by default, so is the Shibboleth proxy credential. The SRB client delegates the Shibboleth proxy credential to the SRB server to request an access to a data resource. The SRB server-side has been extended to parse the user attributes in the new Shibboleth proxy credential to obtain the VO role of the user.

3.4.1. Client-side Implementation

The SIP utilizes the Java GridShib SAML tools [73], capable of handling SAML attribute requests and responses. As shown in Figure 12, the SIP invokes the SAML attribute query client and passes the user proxy credential to it. The attribute query client then contacts Shibboleth at the designated URL. Shibboleth retrieves the user's attributes from OMAR and returns them in the form of a SAML assertion to the SIP. The SIP also utilizes the SAML X.509 binding tool to bind the SAML assertion as a non-critical extension to the user proxy credential. The new credential is called Shibboleth proxy credential.

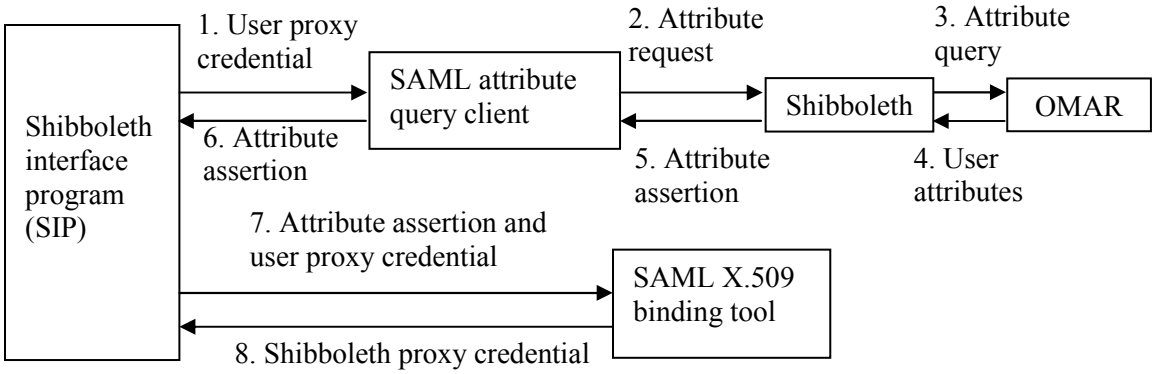

Figure 12: Data flow involving the Shibboleth interface program (SIP)

Certificate:
Data:
Version: 3 (0x2)
Serial Number: 867284 (0xd3bd4)
Signature Algorithm: md5WithRSAEncryption
Issuer: O=Grid, OU=GlobusTest, OU=simpleCA-xxxx.cs.wright.edu, OU=cs.wright.edu, CN=uuuu, CN=613767748
Validity
Not Before: Apr 30 00:16:16 2008 GMT
Not After : Apr 30 12:21:16 2008 GMT
Subject: O=Grid, OU=GlobusTest, OU=simpleCA-xxxx.cs.wright.edu, OU=cs.wright.edu, CN=uuuu, CN=613767748, CN=67935642
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (512 bit)
Modulus (512 bit):
00:8d:cd:e0:1f:b9:57:dc:a9:3a:78:0a:4d:03:64:
2c:94:42:c6:ca:2f:33:b1:71:9f:49:1c:29:2d:73:
dd:8f:1e:10:5d:c0:ef:26:dc:ce:fc:18:9c:f9:60:
e4:11:5a:38:7d:76:63:ec:34:50:b0:e9:38:af:5f:
b0:21:a4:2c:a7
Exponent: 65537 (0x10001)
X509v3 extensions:
1.3.6.1.4.1.3536.1.222: critical
0.0
..+.....
1.3.6.1.4.1.3536.1.1.1.10:
<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" AssertionID="_1063ef80756c76c747609152bbb4b567" IssueInstant="2008-04-30T00:21:16.283Z" Issuer="https://idp.example.org/shibboleth" MajorVersion="1" MinorVersion="1"><Conditions NotBefore="2008-04-30T00:21:16.283Z" NotOnOrAfter="2008-04-30T00:51:16.283Z">
<AudienceRestrictionCondition><Audience>https://globus.org/gridshib</Audience><Audience>urn:mace:gridshib:metadata:sp</Audience></AudienceRestrictionCondition></Conditions><AttributeStatement><Subject><NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName" NameQualifier="https://idp.example.org/shibboleth">CN=uuuu,OU=cs.wright.edu,OU=simpleCA-xxxx.cs.wright.edu,OU=GlobusTest,O=Grid</NameIdentifier></Subject>
<Attribute AttributeName="urn:mace:dir:attribute-def:role" AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">
<AttributeValue>manager</AttributeValue></Attribute>
<Attribute AttributeName="RolePermissions" AttributeNamespace="urn:mace:shibboleth:1.0:attributeNamespace:uri">
<AttributeValue>Permission:1:Resource:DataSet1</AttributeValue><AttributeValue>Permission:1:Action:read</AttributeValue>
<AttributeValue>Permission:2:Resource:DataSet2</AttributeValue><AttributeValue>Permission:2:Resource:DataSet3</AttributeValue>
<AttributeValue>Permission:2:Action:read</AttributeValue>><AttributeValue>Permission:2:Action:write</AttributeValue></Attribute>
</AttributeStatement></Assertion>

Figure 13: Example Shibboleth proxy credential

Figure 13 shows an example Shibboleth proxy credential which contains a SAML assertion issued by Shibboleth. The assertion, italicized in Figure 13, is valid for a limited period of time specified by the *Conditions* statement in the assertion. The Shibboleth proxy credential is stored as a default temporary file, which can be used by any Data Grid applications, including the SRB client.

3.4.2. Server-side Implementation

The SRB server has been modified to obtain the VO role of the user from the Shibboleth proxy credential. The SRB server parses the SAML assertions to obtain the user role based on the algorithm shown in Figure 14.

```
Input: Shibboleth proxy credential of a user
Output: User role for accessing a resource
Method:
    Get the delegated credential from the client
    foreach  certificate in the credential
      Get the extension of the certificate
       if non-critical extension exists then get the SAML assertions

      foreach SAML assertion
          parse the SAML assertion and obtain the attribute name, value and the
namespace
      foreach  attribute
        if the attribute name refers to the user  role
                then return the attribute value representing the role name
        else return the identity of the user
```

Figure 14: Extraction of the user role from the Shibboleth proxy credential

The SRB server tries to retrieve the user's certificates from the credential

presented. If a certificate exists, then the server tries to obtain the non-critical extension containing the SAML assertions from it. The Object Identifier (OID) of that non-critical extension is 1.3.6.1.4.1.3536.1.1.1.10. So, if a non-critical extension with that OID is present, then the server obtains the SAML assertions present in it. From the SAML assertions, the server obtains the attributes of the user, such as the user's role name, and uses them to perform the role-mapping (from a VO role to a local role) and to make all the authorization decisions. If the server could not obtain the SAML assertions from the credential, then it performs the matching between the user's distinguished name (DN) present in the certificate and the DN stored in the MCAT database.

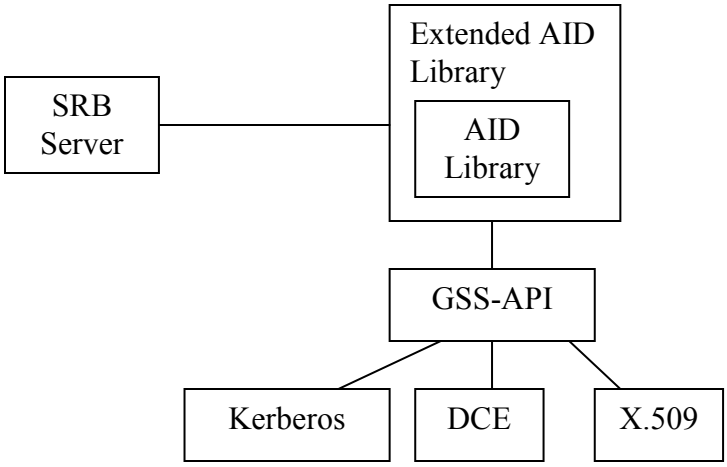

Figure 15: Server-side modifications

Our server-side implementation, shown in Figure 15, extends the San Diego Supercomputer Center (SDSC) Authentication/Integrity of Data (AID) library which integrates the Grid Security Infrastructure (GSI) into the SRB system. The AID library provides an API to the underlying Generic Security Services API (GSSAPI) package, shielding the application programmer from the complexities of the GSSAPI. It also

supports Kerberos and the Distributed Computing Environment (DCE) security via GSSAPI.

3.5. Performance Analysis

We implemented our proposed RBAC system and analyzed its performance in terms of the overheads incurred. Our implementation is compared with the existing implementation of SRB, which does not use Shibboleth for access control. For our implementation, we installed an MCAT-enabled SRB (MES) server on a SuSE Linux machine with a 2.6 GHz Intel Pentium IV processor and 1 GB RAM. SRB version 3.4.2 is used, and Oracle 9.0 is used to manage the MCAT database. To use the GSI, we installed Globus Toolkit 4.0.2. To serve the user requests, a non-MES server is installed on another SuSE Linux machine with a 1.6 GHz Intel dual-core processor and 2 GB RAM. Shibboleth identity provider (IdP) 1.3c was configured to run on the SSL-enabled Apache 2.2.0 and Tomcat 5.0.28 servers. We used OMAR 3.0 beta 1 as the repository for storing the XACML policies. Our MES and non-MES servers are connected by a Fast Ethernet LAN. For the purpose of analysis, the *Sput* command was invoked on the non-MES server, which is an SRB command line utility for importing a local file/directory at a site into the SRB system.

3.5.1. Profiling Details

For profiling the time taken to receive the assertions from the Shibboleth server and to embed them in the user credential a Java method *System.currentTimeMillis*() is

used to get the current system time in milliseconds. Also, for the SRB system analysis, a C time function *gettimeofday*() is used to get the time in the order of microseconds. For more accuracy, the servers and containers were shutdown and restarted before each client invocation of the *Sput* command, as well as before each invocation of the assertion request to the Shibboleth server respectively, to minimize the caching effects.

The main changes are the way the server obtains the VO role from the client's delegated credential and the credential being delegated by the client. So only the GSI security aspects of the client and the server are profiled and analyzed. SRB system supports two GSI authentication settings, GSI authentication and GSI delegation. In case of GSI delegation, the proxy credential of the user is delegated to the server and the server may contact other services on user's behalf. In case of GSI authentication, only the user identity is used for establishing the connection. The extended AID library contains the server side operations performed, such as extracting and parsing the credentials, when the authentication mode is set to GSI delegation.

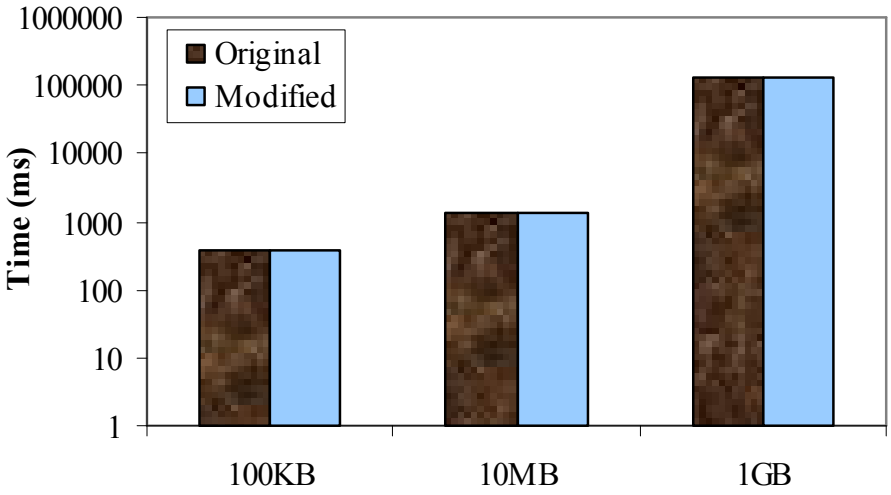

Figure 16: Total execution time for the *Sput* command

The calls made to the original implementation of the SRB server using the user's proxy credential (referred to as *Original* in the figures) and the modified implementation of the SRB server using the user's Shibboleth proxy credential (referred to as *Modified* in the figures), respectively, are profiled. In case of using the Shibboleth proxy credential, an overhead for invoking the SIP is incurred. The SIP obtains the attributes of the user, such as a VO role, in the form of an assertion from the Shibboleth, and then embeds the assertion into the user's proxy credential. The total time for this process is approximately 6 seconds, and the default life time of the user's proxy credential is 12 hours. However, this overhead is incurred only once before the requests are made by the user to the SRB server, and the user can make any number of requests before the proxy credential expires.

The *Sput* command was executed to transfer the data files of different sizes from our non-MES server to the MES server. Figure 16 shows the total times taken for 100 KB, 10 MB and 1 GB data files in the original and modified SRB implementations. We can see that our RBAC system adds a very small overhead, compared to the total execution time required by the original SRB implementation. And the overhead is independent of the data file size. To analyze the overhead further, we profiled the security functions on the client-side and server-side of the SRB system, and the results are described in the following sections.

3.5.2. Client-side Security

To analyze the overhead at the client side, the *Sput* command for transferring a 100 KB data file is used, and the execution times of the following functions of the AID library are measured:

1. The *aid_setup_creds* function sets up the user's proxy credential on the client-side.

2. The *aid_establish_context_clientside* function performs the mutual authentication on the client-side.

The authentication mode is set to GSI delegation, and as shown in Figure 17, the time for setting up the credentials and the time for establishing the context for mutual authentication are almost the same in both implementations. The reason is because all the security functions on the client-side remain unchanged in our modified implementation, except for using the Shibboleth proxy credential instead of the user proxy credential.

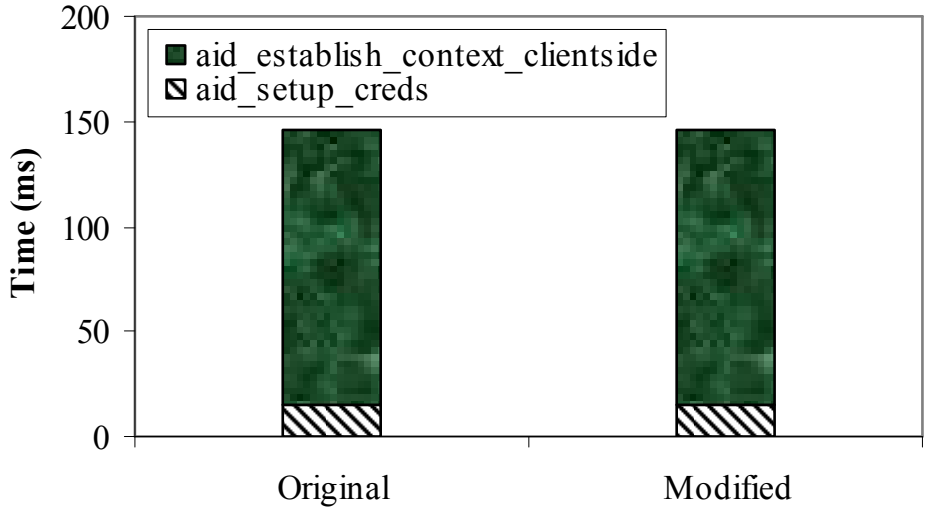

Figure 17: Security overheads on the client-side

3.5.3. Server-side Security

On the server-side, the authentication mode is also set to GSI delegation, and we measured the execution times of the following functions:

1. The *aid_setup_creds* function sets up the server's credential.
2. The *aid_establish_context_serverside_with_delegate* performs the mutual authentication on the server-side and extracts the user's VO role or DN from the Shibboleth proxy credential delegated to the server.

As shown in Figure 18, the execution time of the *aid_setup_creds* function remains the same because there is no change in the server credential. The *aid_establish_context_serverside_with_delegate* function takes a little bit more time in the modified implementation, because of the additional time for parsing the user attributes in the Shibboleth proxy credential and extracting the VO role. However, the

time difference is in the order of a few milliseconds, which is negligible compared to the total time taken for a typical SRB application, like the *Sput* operation.

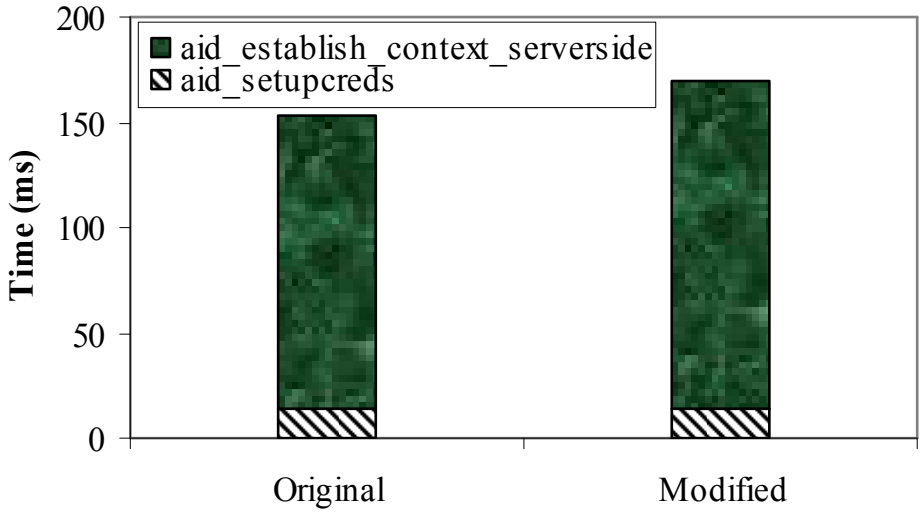

Figure 18: Security overheads on the server-side

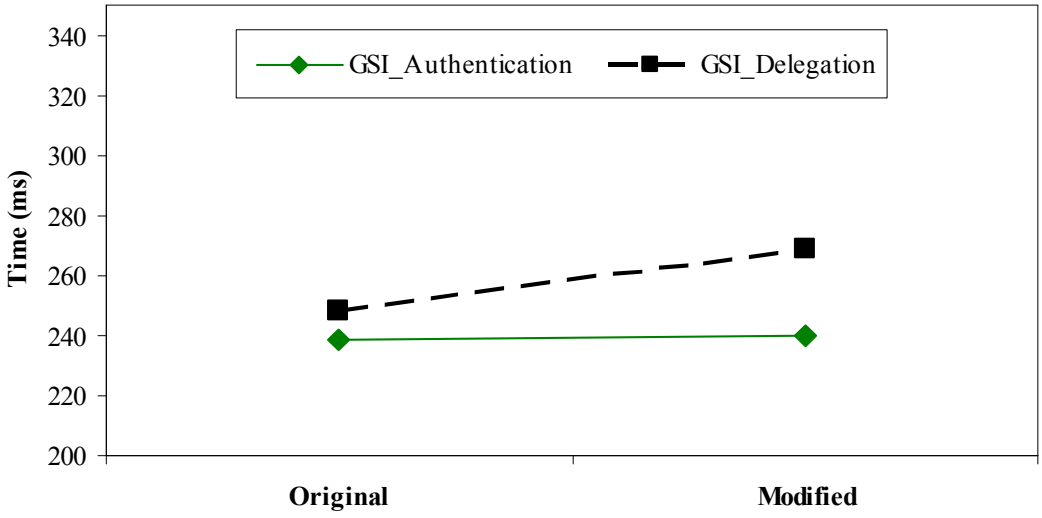

Figure 19: Total time taken for the execution of the client program

Figure 19 shows the total time taken for executing the *Sls* Command with the following authentication settings: GSI authentication and GSI delegation. The *Sls* command was executed with these settings on the original SRB implementation as well as the modified SRB implementation. In case of GSI delegation, the time taken is even higher because in addition to setting up the client's credential there is an additional overhead on the SRB server side. The overhead on the server side is because of the time taken for extracting the SAML assertions from the client's delegated credential and for parsing the assertions to obtain the client's role. In case of GSI authentication, the time taken for the modified implementation is almost same as the original implementation as there is no overhead involved for extracting client's credentials.

3.6. Summary

We enhanced the access control mechanism of the SRB by using Shibboleth, XACML, and OMAR to support role-based access control (RBAC) and the distributed administration of the access control policies. The proposed RBAC system allows quick and easy deployment, and provides privacy protection for the users. Furthermore, the users can be dynamically granted memberships on the VO roles. The administration overhead of the resource providers is reduced because the information about the VO roles and their mapping to local roles is maintained in the MCAT database, so the resource providers do not need to maintain the information about individual users and their access privileges. The specification of access control policies at the VO level eliminates

unnecessary authentication, mapping and connections by denying invalid requests at the VO level.

As our RBAC system uses the push model, the SRB server does not need to contact the attribute service to obtain the user information for each access request. So, it is more efficient in terms of performance when we have a large number of user requests. In Data Grids, as users and resources are dynamic, administrative scalability is also critical. As users join/leave the organization, the administrator may have to update the policies as well as the necessary permissions of individual users. Considering the frequency of these changes, our RBAC system has good administrative scalability because the administrator just needs to grant/revoke the role memberships of the users when they join/leave the system.

Our performance analysis shows that the proposed RBAC system incurs a small overhead for authorizing the user. However, this overhead is quite acceptable when we consider the benefits of our system, such as the scalability in terms of the number of users, reduced administration overhead of the resource providers, and the distributed administration of the access control policies.

4. SEMANTIC-BASED ACCESS CONTROL USING ONTOLOGY AND XACML

Several improvements to the current OGSA-DAI's access control method have been proposed [60, 61]. In [60, 61], we described how the Community Authorization Service (CAS) [58] can be used to RBAC in the OGSA-DAI framework. In [44, 94], we described how Shibboleth [88] and eXtensible Access Control Markup Language (XACML) [52] can be used for specifying the RBAC policies for distributed administration. Shibboleth is an attribute authorization service and is designed to provide user attributes to the resources for access control, and it mainly targets the Internet-based resources [88]. XACML is a standard of the Organization for the Advancement of Structured Information Standards (OASIS) for describing the access control policies uniformly across different security domains [52]. We used the Core and Hierarchical RBAC profile of XACML [52] to specify the access control policies for multiple VOs.

However, all these approaches have two major problems. The first problem is that different organizations may use different terminologies in describing their data resources and user roles. For example, different agencies, such as FBI and CIA, might use different terms to describe the same role: FBI may use the term *agent*, whereas CIA may use the term *field_agent*. These agencies will fail to match the relevant information when a specific term used in the request is different from those used in the data resources and access control policies, despite having the same meaning [14]. The second problem is that the access control policies should be specified for individual data resources. Thus, the management of access control policies may not be scalable in terms of the number of data resources.

To solve these two problems, we propose a new semantic-based access control system using ontology and XACML policy. By using ontology, VOs can resolve the differences in their terminologies, and access control policies can be specified based on concepts, instead of individual data resources and users. For example, we can specify a policy like "agent can read criminal DB," instead of specifying similar policies for all different types of data resources and users. Thus, access control decisions can be made based on the semantics of the users and data resources, rather than just using the syntax of the user's requests and the corresponding access control policies.

Our system is scalable in terms of the number of data resources as well as users, because they can be abstracted based on the concepts, and the access control policies can be specified on the concepts, instead of the individual data resources and users. When new data resources are added, they could be categorized into existing concepts automatically, thus we can reduce the administration overhead.

Our proposed access control system uses XACML to specify the RBAC policies for the data resources hosted by OGSA-DAI. Each XACML policy specifies the actions that can be performed by a VO role on certain resource. A user's request is evaluated by our access control system against the XACML policies based on the attributes of the user, requested resource and action. Currently, there are no tools available for creating, updating and managing XACML policies. We also developed an XACML policy administration tool with those capabilities, and it can automatically generate XACML policies by importing existing RBAC policies.

We implemented our system by modifying the OGSA-DAI server to obtain the user's attributes from the Shibboleth identity provider (IdP) and evaluate the user access request by using the XACML policy and the ontology. We used World Wide Web Consortium (W3C) standard Web Ontology Language (OWL) to represent the ontology of the data resources and users.

Our performance analysis shows that the proposed system incurs a small overhead for obtaining the semantic information and for evaluating the access request against the XACML policy. This overhead is quite acceptable when we consider the benefits of our system, such as scalability in managing the access control policies and reduced administration overhead for the resource providers. Additionally, as the access requests are evaluated before a connection is established to the resource, unauthorized requests can be denied without establishing a connection to the resource.

This chapter is organized as follows: Section 4.1 contains background information. In Section 4.2, we describe the related work in semantic access control. In Section 4.3, we present our proposed semantic-based access control system in OGSA-

DAI using Shibboleth, XACML and ontology. In Section 4.4, we describe the implementation details of the system. In Section 4.5 we present the results of performance analysis. Section 4.6 contains some conclusions.

4.1. Background

4.1.1. Open Grid Services Architecture - Data Access and Integration (OGSA-DAI)

OGSA-DAI is a widely used middleware infrastructure that facilitates uniform access to data resources using a service-oriented architecture (SOA) [3]. Recently, OGSA-DAI has been linked with the Web Services Resource Framework (WSRF). The newly defined OGSA-DAI data service can be dynamically configured and can expose multiple data resources, each of which can be any entity that acts as a source and/or a sink of data [3]. The OGSA-DAI data service provides activities that a client can use to perform data resource manipulation, data transformation and delivery actions on data resources such as relational databases, XML databases, files, etc. The user can specify the operations he/she wishes to perform on a resource in an XML document, called the *perform* document.

OGSA-DAI uses the Grid Security Infrastructure (GSI) for authentication and secure communication. GSI authenticates the user by using his/her proxy credential, which is generated based on the user certificate and is valid for a certain time period (e.g., 12 hours by default). Once the authenticity of the user is verified, the OGSA-DAI data service performs user authorization.

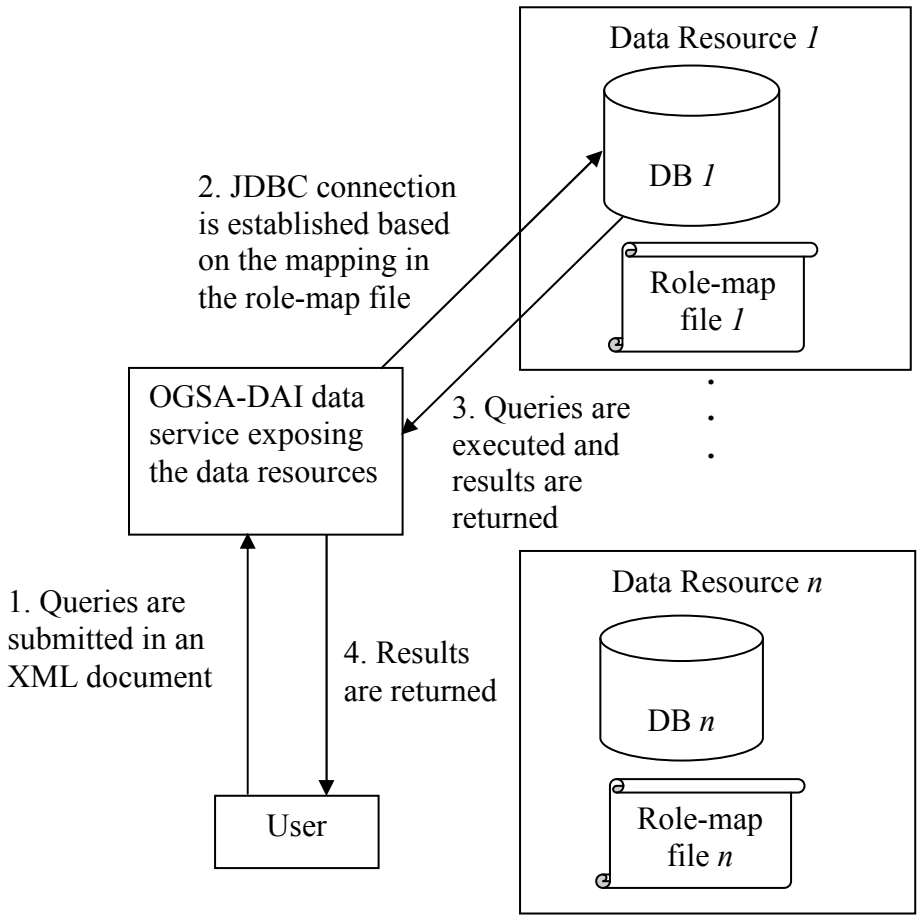

Figure 20: Accessing a data resource through OGSA-DAI

Figure 20 shows how an authenticated user can access a data resource named 'Data Resource *1*' using OGSA-DAI. For each data resource, a role-map file is maintained which contains the information for mapping a Grid user identity to a username and a password that are used to connect to the database at a particular authorization level. The user can submit the queries (in a *perform* document) to the OGSA-DAI data service along with the data resource he/she wishes to access. The OGSA-DAI data service then retrieves the role-map file corresponding to that data resource and maps the user to the local database username and a password based on his/her distinguished name. A Java Database Connectivity (JDBC) connection is then

established between the OGSA-DAI data service and the local database. After the user queries are executed on the local database, the results are returned to the user.

Thus, in the current implementation of OGSA-DAI, the access control information needs to be stored for each and every resource. 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 control information of the current users change. Also, there is no mechanism to classify the data resources in terms of the semantic concepts, thus it is difficult to specify access control on similar data resources.

4.1.2. Shibboleth and GridShib

Shibboleth [10] is an attribute authorization service developed by the Internet2 community for cross-organization identity federation [88]. Shibboleth creates a distributed authorization infrastructure for Web resources, and simplifies the access control policies and their management [4]. Shibboleth provides the user attributes, such as role name and affiliation, to the requested resources by using the Security Assertion Markup Language (SAML) [47] attribute statements. SAML can be used to uniformly express authentication and authorization assertions between different security domains. The resource providers can make authorization decisions by using the user attributes, unlike the case of identity-based authorization. For example, only graduate students studying computer science in a particular university and enrolled in a particular course can gain access to a certain database.

Shibboleth leverages each organization's existing local access control and identity management systems. Shibboleth supports single sign-on and protects the privacy of the users by managing the user attributes and performing user authentication at the user's home institution. By using Shibboleth, the user attribute information is managed by the home institution alone, not by the individual resource providers. The user is authenticated by his/her home institution and the user attributes are provided to the requested resource providers within and outside the home institution. And the users do not have to maintain the accounts at different resources, as their role in the home institution determines their permissions on the resources [77].

Shibboleth allows pseudonymous interaction between users, thus protecting individual privacy while still providing basic security for the resource providers [88]. The Shibboleth IdP 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 [28]. A user could be issued a set of pseudonym identities [88], and he/she could be authenticated at different sites with different identities. The information that binds the set of pseudonym identities to the handle should be maintained securely and can be used when security violations occur.

GridShib [88] is a software system that interfaces Shibboleth and Grid services. For this purpose, it provides two interfaces, one for the Grid services (GridShib-data service interface) and the other for Shibboleth (GridShib-Shibboleth interface). Through

these two interfaces, the request for user attributes is submitted to Shibboleth, and the SAML attribute statement is returned to the Grid service.

4.1.3. Semantic Web Technologies

The Semantic Web is an evolving extension of the World Wide Web in which the semantics of information and services on the Web is defined, making it possible for the Web to understand and satisfy the requests of people and machines to use the Web content [7]. The Semantic Web provides a common framework that allows data to be shared and reused across diverse domains. Semantic Web technologies can be used to resolve the heterogeneity in terminology when the data in various locations and various formats are integrated.

One of the basic components of Semantic Web is ontology. Ontology is an explicit and formal specification of a conceptualization [25], and we can create an ontology to describe the concepts and the relationships between them in a certain domain. Ontology consists of classes (or concepts), relations, instances of classes, and axioms; hence it can be defined as a 4-tuple $<C, R, I, A>$, where C is a set of classes, R is a set of relations, I is a set of instances, and A is a set of axioms [14].

Each resource on the Semantic Web involves metadata. Resource Description Framework (RDF) [66] is a W3C standard for the notation of metadata descriptions of the Web-based resources. RDF Schema (RDFS) [67] was subsequently developed so that structured information could be represented using RDF syntax. RDFS is based on the concept of class and inheritance, and it features simple constructs for describing

ontologies. The W3C standard for describing ontologies on the Semantic Web is the Web Ontology Language (OWL) [56], which supports richer semantics on top of RDFS. OWL is a semantic markup language for publishing and sharing ontologies on the Web, and it provides mechanisms for creating all the components of an ontology: classes, instances, relations and axioms. OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

Integrated tools, such as Jena [31], Sesame [76] and Pellet [59], are available for using RDF, RDFS and OWL. They provide inference capability as well as query languages, such as SPARQL [79], for querying the RDF stores. Simple inference is possible in RDFS and OWL based on inheritance. More complex rules require the use of rule languages. Rule languages allow users to write rules to reason about individual entities and to infer new knowledge about them. Several rule languages, such as Semantic Web Services Language (SWSL) [81], Rule Markup Language (RuleML) [69] and Semantic Web Rule Language (SWRL) [27], are available. The rule language part of SWSL is based on the frame logic and high-order logic programming. RuleML permits both forward-chaining (bottom-up) and backward-chaining (top-down) inference methods for deduction, rewriting, and further inferential and transformational tasks [69]. SWRL is a combination of the sublanguages of OWL with the sublanguages of RuleML. Our proposed architecture is based on OWL and SWRL.

4.2. Related Work in Access Control

Grid environments are formed in a dynamic manner, where both users and data resources may join or leave the VO in an ad-hoc manner. Traditional access control models are not efficient enough to deal with the Data Grids with a huge amount of heterogeneous data resources, where new data resources are incorporated to the system dynamically. Moreover, different data resources may need different access control policies, and those policies may change frequently. To address these issues, several new access control methods have been proposed for Data Grids [44, 61], but not many of them are using semantics.

The semantic access control (SAC) model [90, 91] uses different layers of metadata to take advantage of the semantics of the clients, resources, context and attribute certificates. They used semantic policy language (SPL), which is an XML-based language, to specify the access control policies for Web Services. However, SPL does not provide powerful metadata representation and reasoning capabilities [62].

In [75], the authors propose the use of semantic access certificates for medical applications in Grid environments. The semantic access certificates contain the access control policies based on the roles of the user and the semantics of the stored data. However, this method is focused on medical applications, and they did not use a standard for the representation of metadata.

KAoS [85] is a policy and domain services framework which uses OWL to represent authorization policies. However, using OWL as the basis for representing and reasoning about access control policies in Data Grids may not be efficient because OWL

is not designed for the specification of access control policies. Also, the gap between the specification and the actual implementation of such policies cannot be coped with automatically [84]. Another issue is that KAoS is deployed as an independent Grid service, so a client needs to register with that service in order to access the resources [86].

Rei [35] is a policy framework which integrates policy specification and reasoning. The policy specification language of Rei (version 2.0) is based on OWL-Lite and uses a rule-based approach to specify access control policies as constraints over the actions on resources [68]. The rule-based approach enables the definition of policies that refer to a dynamically determined value, thus providing flexibility in policy specification. The policy engine of Rei reasons over the policies and the domain knowledge in RDF and OWL. However, the rules have a specific syntactic form, so they should be processed differently from OWL [83].

In [13], the authors proposed to incorporate ontology-based descriptions of resources and subjects into XACML policies. However, they used RDFS which supports less semantics than OWL. In [62], the authors used an extension of the XACML standard to simplify the specification and maintenance of attribute-based access control policies by providing an ontology-based attribute management system. However, they did not include actual resources and the policy enforcement point in their scenario.

In [57], the authors presented Semantic Access Control Enabler (SACE) which is based on RBAC. It allows the user to query the database by using the terms other than those specified in the database schema. The role of the user can also be specified by using a term not present in the role-hierarchy. It uses OWL to represent the ontology. However,

this approach cannot be deployed where the mediator service cannot be trusted, and it is mainly based on converting the query attributes.

4.3. Semantic-based Access Control Using Shibboleth, XACML and Ontology

We propose a semantic-based access control system for data resources in OGSA-DAI by using Shibboleth, XACML and ontology. The proposed access control system makes the authorization decisions based on the user's request, the access control policy specified in XACML, and the semantic mappings/relationships represented in OWL. The original implementation of OGSA-DAI uses the Grid user identity for access control, which clearly does not support privacy protection. With our system, the user does not need to send the user identity and can control what information is released by the Shibboleth IdP to the resource providers. In some cases, the resource provider may only need to know that the user is a member of an organization in order to make the access control decision. In that case, only the membership information will be released to the resource provider. Thus, the resource provider is given only as much user information as needed to make the access control decision, and the privacy can be protected.

4.3.1. eXtensible Access Control Markup Language (XACML)

XACML is a widely used standard for representing attribute-based access control policies. The XACML specification [52] defines both a language model and a data-flow

model. The XACML language model defines the following main components to represent policies:

(1) A <PolicySet> contains a set of access control policies or other policy sets.

(2) A <Policy> represents an access control policy described through a set of rules.

(3) A <Rule> represents an access rule or permission. The effect of a rule is either *permit* or *deny*.

An XACML <PolicySet>, <Policy> or <Rule> may contain a <Target> element. A <Target> element specifies a set of subjects, resources, actions and environments to which the <PolicySet>, <Policy> or <Rule> is applied [52].

Figure 21 shows a sample policy in XACML v3.0 which specifies that users with the *administrator* role have *read* and *write* permissions on the *Phonebook* resource.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:schema:os"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  PolicyId="urn:oasis:names:tc:xacml:3.0:example:policyid:policy1"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
 <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:ruleid:rule1" Effect="Permit">
   <Target>
    <AnyOf>
     <AllOf>
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">administrator</AttributeValue>
        <AttributeDesignator Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Match>
     </AllOf>
    </AnyOf>
    <AnyOf>
     <AllOf>
      <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Phonebook</AttributeValue>
        <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:target-namespace"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Match>
     </AllOf>
    </AnyOf>
    <AnyOf>
     <AllOf>
      <Match  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
        <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Match>
     </AllOf>
     <AllOf>
      <Match  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
        <AttributeDesignator Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
         AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </Match>
     </AllOf>
    </AnyOf>
   </Target>
  </Rule>
</Policy>
```

Figure 21: An example XACML policy

4.3.2. Creation of Ontology Using Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL)

OWL can be used to represent all the components of ontology: classes, instances, relationships and axioms [55]. Classes are defined in OWL using *owl:Class* element. For example, a *Customer* class can be defined as *<owl:Class rdf:about="Customer"/>*. Classes in OWL can be organized in a hierarchy based on the superclass-subclass relationships. OWL has two predefined classes: *<owl:Thing>,* which is the most general class, and *<owl:Nothing>,* which is an empty class. The *Customer* class is thus a subclass of *<owl:Thing>* and a superclass of *<owl*:*Nothing>*.

Instances of classes are the actual entities and are referred to as individuals. An instance is created by declaring it as a member of a class. OWL does not have a unique name assumption, i.e., different names may refer to the same individual. For example, Matt and Mathew may refer to the same individual. This ambiguity can be resolved using *<owl:SameAs>* and *<owl:DifferentFrom>*, which assert that an individual is same as or different from the other individual, respectively.

In OWL, there are two main categories of properties regarding the instances: *datatype properties*, which relate instances to *datatype* values, and *object properties*, which relate instances to other instances. Properties can have a specified domain and range. OWL uses a built-in XML schema for *datatypes*, which is referenced using the URI: http://www.w3.org/2001/XMLSchema. The following is an example of a *datatype property*, specifying *numItemsPurchased* is a non-negative integer.

```
<owl:DatatypeProperty rdf:ID="numItemsPurchased">
        <rdfs:range
        rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</owl:DatatypeProperty>
```

The following is an example of an *object property*, which relates the customers to the regions they are located in.

```
<owl:ObjectProperty rdf:ID="locatedIn">
        <rdfs:domain rdf:resource=" #Customer" />
         <rdfs:range rdf:resource="#Region"/>
</owl:ObjectProperty>
```

In OWL, we can define property restrictions to restrict individuals that belong to a class [55]. There are three types of property restrictions: *quantifier restrictions*, such as <*owl:someValuesFrom*> and <*owl:allValuesFrom*>; *cardinality restrictions*, such as <*owl:minCardinality*>, <*owl:maxCardinality*>; and *hasValue* restriction denoted by <*owl:hasValue*>. The following is an example of a restriction, which specifies that for all instances of *OhioCustomers*, if they have a *locatedIn* property, then the location has to be an *OhioRegion*.

```
<owl:Class rdf:ID="OhioCustomers">
        <rdfs:subClassOf rdf:resource="Customer"/>
          <rdfs:subClassOf>
                <owl:Restriction>
                        <owl:onProperty rdf:resource="#locatedIn" />
                        <owl:allValuesFrom rdf:resource="#OhioRegion" />
                </owl:Restriction>
          </rdfs:subClassOf>
```

</owl:Class>

OWL provides additional constructors such as *<owl:unionOf>*, *<owl:intersectionOf>*, *<owl:complementOf>*, *<owl:oneOf>* and *<owl:disjointWith>* to form classes [55]. The *unionOf* defines a class as a union of other classes. The *intersectionOf* defines a class as an intersection of other classes. The *complementOf* defines a class as a complement of other class. The *oneOf* defines a class by enumerating its members, and no other individual can be declared to belong to this class. The *disjointWith* defines that a class is disjoint with other classes. The following example states that the class *Man* is disjoint with the class *Woman*:

<owl:Class rdf:about="#Man">
<owl:disjointWith rdf:resource="#Woman"/>
</owl:Class>

OWL also contains class axioms such as *subClassOf*, *equivalentClasses*, *disjointClasses*, and *disjointUnion*. The *subClassOf* axiom states that one class is a subclass of another class. For example, *Customer* and *agent* classes could be stated as subclasses of *Person* class, as shown in Figure 22. The *equivalentClasses* axiom states that a set of classes are equivalent. The *disjointClasses* axiom states that a set of classes are disjoint. The *disjointUnion* axiom defines a class as a union of other classes, all of which are pair-wise disjoint.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY company "http://xxx/company.owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&company;"
        xmlns:owl="&owl;"
        xmlns:rdf="&rdf;"
        xmlns:rdfs="&rdfs;">
<!-- Ontology Information -->
<owl:Class rdf:about="Phonebook">
            <rdfs:subClassOf rdf:resource="#book"/>
 </owl:Class>

<owl:Class rdf:about="Person"/>

  <owl:Class rdf:about="EmployeePhonebook">
             <rdfs:subClassOf rdf:resource="#Phonebook"/>
  </owl:Class>
<owl:Class rdf:about="Customer">
             <rdfs:subClassOf rdf:resource="#Person"/>
  </owl:Class>

  <owl:Class rdf:about="CustomerPhonebook">
              <rdfs:subClassOf rdf:resource="#Phonebook"/>
  </owl:Class>
…
  <owl:Class rdf:about="agent">
            <rdfs:subClassOf rdf:resource="#Person"/>
  </owl:Class>
  <owl:Class rdf:about="field_agent">
            <rdfs:equivalentClassOf rdf:resource="#agent"/>
             <rdfs:subClassOf rdf:resource="#Person"/>
  </owl:Class>
</rdf:RDF>
```

Figure 22: A part of a company's ontology in OWL

```
<swrl:Imp>
  <swrl:head>
    <swrl:AtomList>
      <rdf:first>
        <swrl:DatavaluedPropertyAtom>
          <swrl:argument2 rdf:datatype="&xsd;boolean">true</swrl:argument2>
          <swrl:propertyPredicate rdf:resource="#wholeSaleCustomer"/>
          <swrl:argument1 rdf:resource="#x"/>
        </swrl:DatavaluedPropertyAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:head>
  <swrl:body>
    <swrl:AtomList>
      <rdf:first>
        <swrl:DatavaluedPropertyAtom>
          <swrl:argument2 rdf:resource="#a"/>
          <swrl:propertyPredicate rdf:resource="#urn:example:numItemsPurchased"/>
          <swrl:argument1 rdf:resource="#x"/>
        </swrl:DatavaluedPropertyAtom>
      </rdf:first>
      <rdf:rest>
        <swrl:AtomList>
          <rdf:first>
            <swrl:BuiltinAtom>
              <swrl:builtin rdf:resource="&swrlb;greaterThanOrEqual"/>
              <swrl:arguments>
                <rdf:Description>
                  <rdf:type rdf:resource="&rdf;List"/>
                  <rdf:first rdf:resource="#a"/>
                  <rdf:rest>
                    <rdf:Description>
                      <rdf:type rdf:resource="&rdf;List"/>
                      <rdf:first rdf:datatype="&xsd;nonNegativeInteger">500</rdf:first>
                    </rdf:Description>
                  </rdf:rest>
                </rdf:Description>
              </swrl:arguments>
            </swrl:BuiltinAtom>
          </rdf:first>
        </swrl:AtomList>
      </rdf:rest>
    </swrl:AtomList>
  </swrl:body>
</swrl:Imp>
```

Figure 23: An example SWRL rule

We used OWL to represent the metadata about the resources and user attributes. For example, the ontology shown in Figure 22 specifies that *EmployeePhonebook* and *CustomerPhonebook* classes are subclasses of *Phonebook* class, so they are inheriting the

properties and relationships of *Phonebook*. Similarly, the relationships between the values of a user attribute, such as user's role or affiliation, can also be specified in an ontology. For example, *field_agent* could be specified as equivalent to *agent*.

For reasoning, in addition to inheritance, complex rules can be specified for classes by using SWRL. For example, Figure 23 includes an SWRL rule stating that if a customer has purchased more than 500 items, then the customer is considered as a wholesale customer.

4.3.3. Proposed Architecture

In our system, Shibboleth manages the user attribute information, grants the users' memberships on VO roles, and then authorizes them to use the privileges of those roles. For example, as shown in Figure 24, Shibboleth can return an SAML attribute which indicates that the role of the user is *administrator*. Thus, the resource provider is given only as much user information as needed to make the access control decision, and the privacy of the user can be protected.

```
SAMLAttribute
{
  name='urn:mace:dir:attribute-def:role'
  namespace='urn:mace:shibboleth:1.0:attributeNamespace:uri'
  value #1 ='administrator@abc.org'
}
```

Figure 24: An example SAML attribute returned by Shibboleth

In traditional systems, an organization's entire security policy is managed by a single central authority; however, in a VO, its security policy is distributed across its

member organizations. Thus, it may be necessary to manage the user attributes across different organizations and to aggregate them for an authorization decision. Shibboleth allows the VO to manage different subsets of its attribute space at different member organizations, as each member organization operates a Shibboleth service. The use of Shibboleth also ensures the privacy of users, as the individual user information is solely administered at the user's home organization and not by the individual resource providers.

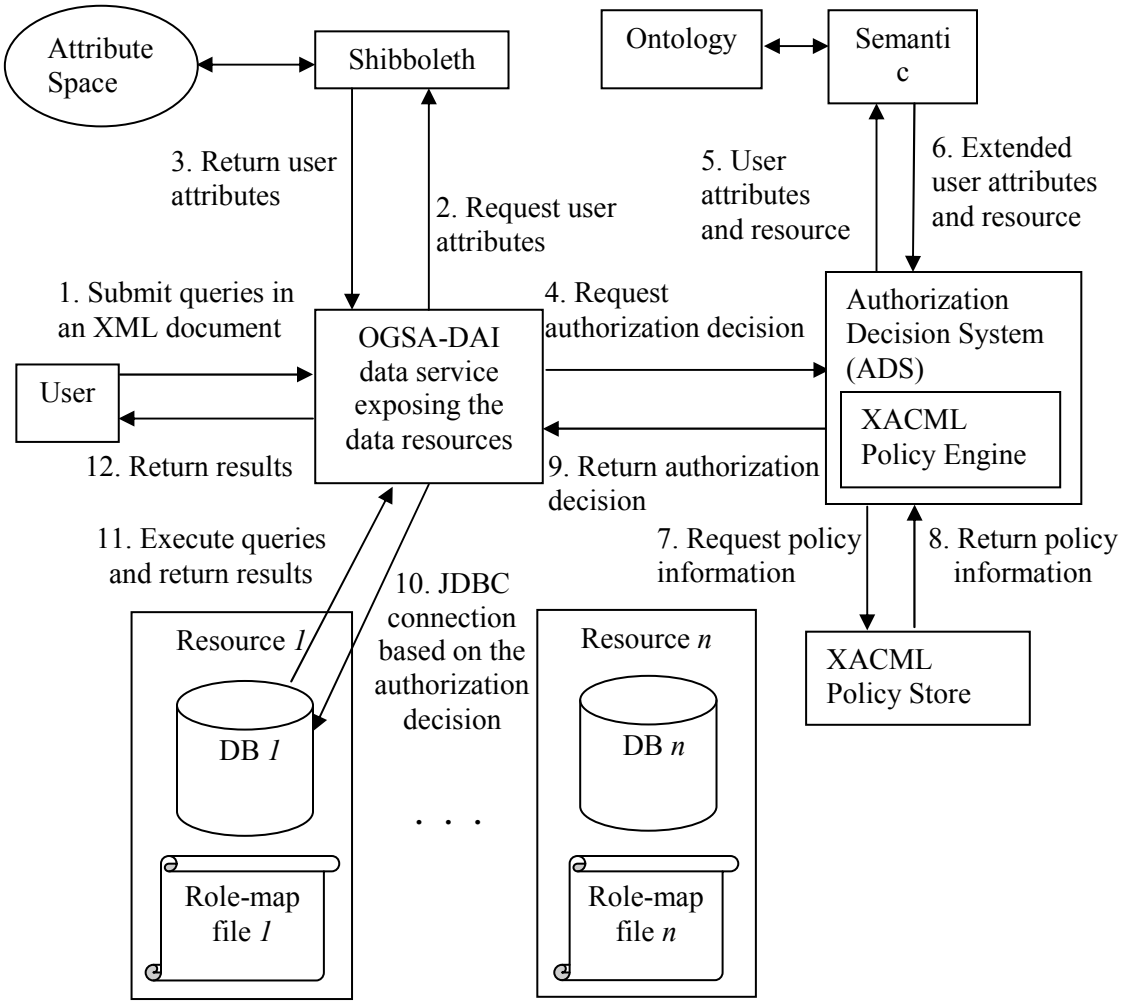

Figure 25: Proposed architecture

Our system uses the metadata about the data resources and user attributes into consideration before making an access control decision. Our proposed architecture is shown in Figure 25. The user contacts the OGSA-DAI data service and submits the *perform* document along with the name of the data resource (e.g., *EmployeePhonebook*) he/she wishes to access. The OGSA-DAI data service then contacts the Shibboleth IdP (through the GridShib interface) to retrieve the user attributes. The OGSA-DAI data service then contacts the Authorization Decision System (ADS) to obtain an access control decision, by sending the user request and user attributes. The ADS retrieves the ontology information of both data resources and user attributes from the semantic interface, which looks up the ontology and invokes an inference engine. The ADS then retrieves the corresponding XACML policy information from the XACML policy store.

```
<Request xmlns="urn:XACML3.vineela.wsu.org">
  <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:3.0:attribute:role">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">administrator</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" includeInResult="true">
      <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">EmployeePhonebook</AttributeValue>
    </Attribute>
  </Attributes>
  <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
    </Attribute>
  </Attributes>
</Request>
```

Figure 26: A user request sent to the XACML policy engine

For decision making, a XACML policy engine [78] is used to evaluate the user request against the XACML policy. The XACML policy engine takes the user request containing the user attributes and the actions on the data resource. An example request

for a user in *administrator* role seeking *read* access to *EmployeePhonebook* resource is as shown in Figure 26.

If the decision cannot be made on the original user request, then ADS modifies it using the semantic information (i.e., mappings) and sends it to the XACML policy engine again for evaluation. For example, if the *administrator* role can perform *read* and *write* on the *Phonebook* (as specified in Figure 21), then the ADS can allow the user with the *administrator* role to perform those actions on the *EmployeePhonebook* because it is a subclass of the *Phonebook*. The ADS returns the decision to the OGSA-DAI data service, which then establishes a connection to the requested data resource.

The data flow in our system is as follows:

1. The user sends the queries (in a *perform* document) along with the data resource name (e.g., *EmployeePhonebook*) to the OGSA-DAI data service.
2. The OGSA-DAI data service then contacts the Shibboleth server at the user's home institution to obtain the user attributes.
3. The Shibboleth IdP retrieves the user attributes from the attribute space and returns the attributes to the resource provider in the SAML format (e.g., attribute-role: *administrator*).
4. To determine whether the user's request is allowed or not, the OGSA-DAI data service requests the Authorization Decision System (ADS) to make a decision.
5. The ADS invokes the semantic interface to obtain ontology information related to the data resource and user attributes.

6. The semantic interface looks up the ontology and performs the inference to obtain the concepts and the corresponding mappings for the data resource and user attributes. For example, *EmployeePhonebook* can be mapped to *Phonebook*. Then, the semantic interface returns the mappings to the ADS.
7. The ADS then retrieves the corresponding access control policy of the data resource from the XACML policy store.
8. The ADS makes an authorization decision based on the access rules specified in the XACML policy and the semantic information pertaining to the user request.
9. The ADS returns *permit* or *deny* to the OGSA-DAI data service.
10. The OGSA-DAI data service establishes a JDBC connection to the requested data resource if the access control decision returned from the ADS is *permit*. Otherwise, the user's request is denied.
11. After the JDBC connection is established, the requested operations are executed on the data resource, and the results are returned to the OGSA-DAI data service.
12. The OGSA-DAI data service then returns the results to the user.

Our access control system is secure because only the user requests that are confirming to the access control policy are authorized. Moreover, all the user requests that are authorized by the original OGSA-DAI (without using semantics) would be authorized by our access control system. That is, our access control system does not impose any additional limitation on the access to the data resources.

Our proposed system uses a single ontology for access control, hence we do not need to resolve the conflict between different ontologies regarding class relationships.

Suppose that an access control policy in our system states that the user with an attribute value V_1 can access a data resource R_1. Then, the user with an attribute value V_2 is allowed to access a data resource R_2 only if V_2 is an equivalent class or a subclass of V_1 and/or R_2 is an equivalent class or a subclass of R_1. This is because if V_2 (R_2) is a subclass of V_1 (R_1) then V_2 (R_2) inherits the properties of V_1 (R_1).

For example, *Phonebook* can be specified as a subclass of *book*, and *manager* can be specified as subclass of *administrator*. In this case, if an access control policy allows an *administrator* to access *Phonebook*, then both *manager* and *administrator* can access *Phonebook* and its subclass *EmployeePhonebook*, but cannot access *book* because *book* is a superclass of *Phonebook*. Thus, the use of ontology in our access control system does not allow unauthorized user requests.

However, a compromised Shibboleth IdP may provide the user attributes such that the user is mapped to a role that allows him/her to access the data resources which are not allowed otherwise. This problem can be avoided by monitoring the log files and checking for illegal access patterns. If a Shibboleth IdP is found to be compromised, then the OGSA-DAI data service can remove it from the list of trusted servers and deny all the user requests originating from the organization hosting the compromised Shibboleth IdP.

4.4. Implementation Details

We implemented our proposed access control system with OGSA-DAI, Shibboleth, XACML, and OWL. We deployed the OGSA-DAI data service in the Globus Toolkit

container. Our proposed system uses the GSI authentication, so if a user tries to access the OGSA-DAI data service using an expired or invalid proxy credential, then the authentication fails and his/her request will be denied.

An example client session is as shown in Figure 27. The user invokes *grid-proxy-init* to create a user proxy credential, which is generated based on the user certificate. The user then contacts the OGSA-DAI data service at the designated URL (https://pc2.cs.wright.edu:8484/wsrf/services/ogsadai/DataService) along with the identity of the data resource (*PhoneBook*) and the *perform* document (fileAccess.xml) containing the actions to be executed on the data resource. In this example, the communication between the client and OGSA-DAI data service uses transport level security and encryption.

```
user@pc1>grid-proxy-init
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-
pc1.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 Aug  5 09:32:50 2009

user@pc1>ant dataServiceClient
-Ddai.url=https://pc2.cs.wright.edu:8484/wsrf/services/ogsadai/DataService
-Ddai.resource.id=PhoneBook
-Ddai.action=examples/Perform/Files/fileAccess.xml -Ddai.tls=encryption
```

Figure 27: Example Client-session

For authorization, our proposed system utilizes the user attributes issued by the Shibboleth IdP. The OGSA-DAI data service contacts the Shibboleth IdP to obtain the attributes of the authenticated user. If the Shibboleth IdP and the OGSA-DAI data service

trust each other, then the Shibboleth IdP releases the user attributes through the GridShib interface [88]. Otherwise, no user attributes are released.

The current implementation uses the pull model, where the user contacts the OGSA-DAI data service that retrieves the user attributes from the Shibboleth IdP. The main advantage of the pull model is that it can be easily deployed and the user does not even need to know that Shibboleth is involved in the decision making. We installed the Shibboleth IdP and deployed it in a Web container. So, any Web service or Web application running on a Web server can contact the Shibboleth IdP on behalf of the client (or user) to obtain the user attributes.

We used Sun's XACML policy engine [80] along with the SICSACML patch [78] to support XACML version 3.0. For the inference on ontology, we used both Jena and Pellet. Jena supports SPARQL better, but it does not support SWRL while Pellet does.

4.4.1. Creation and Administration of XACML Policies

Administration of XACML policies is a difficult task because each XACML policy has several components, as shown in Figure 21, and the number of XACML policies may be very large in Grid environment. However, no efficient tool is available for the creation and update of XACML policies. So, we developed an XACML administration tool and a GUI in Java. Our tool allows the user to specify a RBAC policy, through GUI shown in Figure 28, and then it creates and stores the corresponding XACML policy. It allows the user to update the existing XACML policies, and user can also load existing RBAC policy database into the XACML policy store. Using this tool,

the policy administrator can create new users, roles, resources, and actions. It allows the administrator to change the user-role assignment and the permissions on a role..

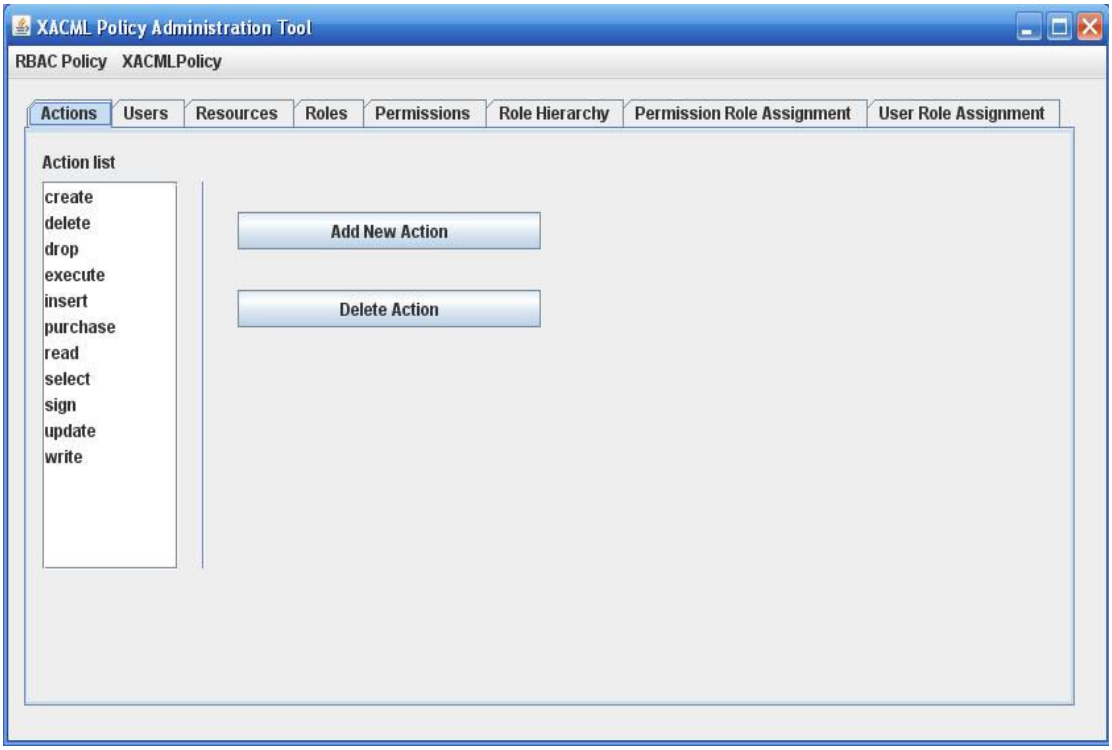

Figure 28: XACML policy administration tool

The Figure 28 shows the graphic interface for creating and managing RBAC policies in XACML. The RBAC policies are converted into XACML format using the core and hierarchical RBAC profile of XACML. For each role, a Role <PolicySet> (RPS) is created. The RPS contains the role name and the id of the corresponding Permission <PolicySet> (PPS). The permissions of a role are represented as rules, and they are contained in the PPS. Each rule specifies the resources and allowed actions on them. For example, the permission *read* on *PhoneBook* resource is represented as a <Rule> as shown in Figure 29.

```
<Rule Effect="Permit" RuleId="Permission:to:write:on:file1">
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Phonebook</AttributeVal
ue>
          <AttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
          <AttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
</Rule>
```

Figure 29: XACML representation of a permission

4.5. Performance Analysis

We have configured the server-side of OGSA-DAI to use Shibboleth. The OGSA-DAI server has been modified to obtain the user attributes from the Shibboleth IdP and communicate with the Authorization Decision System (ADS). The ADS retrieves the

policy information from the XACML policy store and then evaluates the user request to make an authorization decision.

The overheads incurred with our proposed access control system are compared with those of the existing authorization system of OGSA-DAI. Our performance analysis is based on the parameters used for the profiling and benchmarking of the components of OGSA-DAI as described in [30]. Also, as only security aspects are changed by our implementation, we measured the overheads of the security components.

In our system, OGSA-DAI WSRF Release 2.2 was deployed in the Globus Toolkit 4.0.7 container running on a Linux machine with a dual-core 2.8 GHz Intel Pentium processor and 2 GB of RAM. Shibboleth IdP 1.3.3 was configured to run on SSL-enabled Apache 1.2.21 and Tomcat 5.5.23 servers. For the purpose of analysis, we created an ontology containing 272 classes and used a *perform* document containing a request to *read* from a 600 KB text file.

4.5.1. 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 is used to log the time in milliseconds. For more accuracy, the Tomcat, Apache and Globus Toolkit containers are shut down and restarted before each client request, in order to minimize the caching effects within Globus Toolkit, OGSA-DAI and Shibboleth. The main change from the original configuration of OGSA-DAI is the way authorization is performed at

the server-side. Hence, only the security aspects of the client and the server are profiled and analyzed.

Globus Toolkit uses GSI which allows two levels of security: transport-level and message-level. In transport-level security, the entire communication channel between the client and the server is encrypted. In message-level security, only the message is encrypted, so it has flexibility that the message can be transmitted over any transport. The message-level security offers more features than the transport-level security, but it takes more time as each message needs to be encrypted. The performance is analyzed based on these two levels of security established between the client and the server as shown below. For each of these two levels, we recorded the times taken with and without using the Shibboleth and ADS.

1) *TLS*: Enforces GSI secure conversation between the client and the server of OGSA-DAI with transport-level security and encryption.

2) *MLS*: Enforces GSI secure conversation between the client and the server of OGSA-DAI with message-level security and encryption.

4.5.2. Client-side Security

A call is made to the OGSA-DAI data service for each of the two security levels with and without using the Shibboleth and ADS. In case of using the Shibboleth and ADS, additional overhead exists for contacting the Shibboleth IdP, retrieving user attributes, and evaluating the user request against the XACML policy by utilizing ontology. The *getVersion* method of the data service returns the version of OGSA-DAI

used. The *getResourceIDs* method of the data service returns the list of resources hosted by the data service. The *perform* method of the data service takes the *perform* document, which contains the user query, and returns the result 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. A call to the *setConnection* method to establish the connection with the data service.
2. Calls to the *getVersion, getResourceIDs* and *perform* methods.

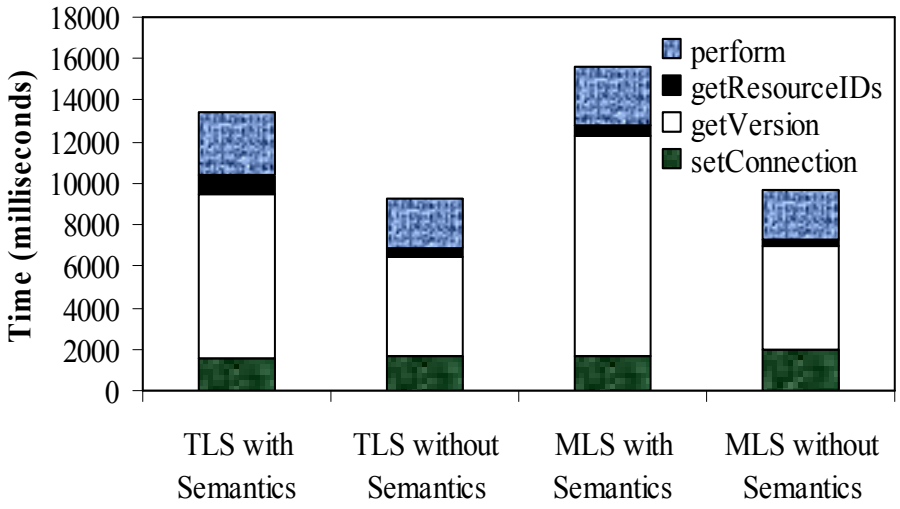

Figure 30: Client-side security

The corresponding times are shown in Figure 30, and as observed, the time taken by the *setConnection* method is almost the same. It involves connecting to the data service at the given URL, retrieving its information, and determining the specification of

Web services used for the OGSA-DAI distribution. As this process does not require any authorization, it takes almost the same time regardless of the security enforced by the data service. The calls to the *getVersion, getResourceIDs* and *perform* take longer when the Shibboleth and ADS are used because the user attributes need to be retrieved from the Shibboleth IdP, and then the user request is evaluated to determine whether the user is authorized to perform the requested operation. When the Shibboleth and ADS are not used, authorization is performed using a role-map file which maps the Grid user identity to a local user account.

The times for executing the *getVersion* method are longer compared with other methods, because it takes time to retrieve the user attributes from the Shibboleth IdP. The later calls take less time as they may require the same user attributes that are obtained previously and cached by the data service.

4.5.3. Server-side Security

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

1. Times taken for retrieving user attributes from the Shibboleth service during the *getVersion*, *getResourceIDs* and *perform* method calls.

2. Time to evaluate the request by using ontology and XACML policy.

3. Time taken for the *perform* operation.

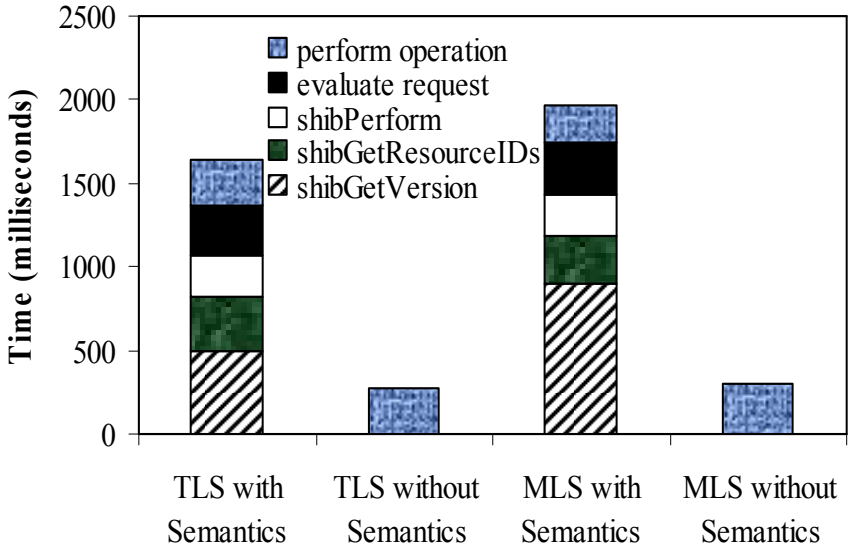

Figure 31: Server-side security

Figure 31 shows the corresponding times taken for the OGSA-DAI data service with and without the Shibboleth and ADS. The data service with the Shibboleth and ADS contacts the Shibboleth IdP to obtain the user attributes during the *getVersion*, *getResourceIDs* and *perform* method calls, and the corresponding times are denoted by *shibGetVersion*, *shibGetResourceIDs* and *shibPerform* in Figure 31. The user attributes obtained to make a call to *getVersion* method can be cached by the data service, and then reused by the same client, for the subsequent calls to *getResourceIDs* and *perform* methods. So, it takes less time for those methods.

When the Shibboleth and ADS are not used, this overhead is not incurred as the data service does not use the user attributes for authorization. Instead, the server obtains the user's Grid identity, known as distinguished name (DN), and maps it to a local user account using a role-map file. To retrieve the user's DN, the server tries to obtain the user's proxy credential using the transport-level security mechanism. If that fails, then the

server tries to obtain the user's DN using the message-level security mechanism. If these two attempts are unsuccessful, the server determines that the client does not have any security context setup, but may allow anonymous access based on "No Certificate Provided." The mapping of a DN to a local user account usually takes less than a millisecond, so it is not shown in Figure 31.

The data service with the Shibboleth and ADS authorizes the user request by evaluating it using the XACML policy and ontology. The time required for this evaluation is usually very small, as shown in Figure 31. The time taken for the actual *perform* operation is the same for the data services with and without using the Shibboleth and ADS.

4.6. Summary

We proposed a semantic-based access control system for OGSA-DAI by using Shibboleth, XACML and ontology to reduce administration overhead and to achieve greater interoperability in Data Grids. By using the established XACML standard, we can represent the access control policies of different VOs uniformly. And, by using semantics for access control, the permissions can be associated with the concepts, instead of individual objects, thus resolving the differences in terminologies and reducing the burden on the administrator. We also developed an XACML policy administration tool for creating the XACML policies from existing RBAC policies and managing them.

Our performance analysis shows that the proposed access control system incurs a small overhead in evaluating the user request. However, this overhead is quite acceptable

when we consider the benefits of our system, such as the scalability in terms of the number of users and data resources and reduced administration overheads of the resource providers. For a complex query on a large database, this overhead is very small compared to the execution time of the *perform* document.

5. CONCLUSION

In this dissertation, we developed a semantic and role-based access control system for Data Grids. Our proposed access control system resolves certain issues, such as scalability, privacy protection, principle of least privilege and interoperability. Our proposed systems are based on role-based access control and hence access control decisions are based on user roles instead of user identities, thus improving the scalability. By using Shibboleth, our system protects the privacy of the users because Shibboleth allows the user attributes to be maintained by the home institution alone and not by individual resources. Our role-based access control system supports the principle of least privilege, i.e., the users are given only those privileges required for completing a certain set of tasks.

Next, our semantic-based access control system resolves two major problems. The first problem is that different organizations may use different terminologies in describing their resources and user roles. The second problem is that the access control policies should be specified for individual data resources. Our systems allow quick and easy deployments, and supports privacy protection of the users. Our performance analysis shows that the systems add a small overhead to the existing authorization infrastructures

of OGSA-DAI and SRB while improving the scalability and reducing the administration overhead.

6. FUTURE WORK

Existing databases store XACML policy as an XML file or object. Storage, administration, and management of XACML policies, creates unique demands that are not addressed, including resolving the association between the policies. Creation of XACML policy is itself a significant challenge for administrators, as there are no tools available for creation and verification of these policies. Additionally, as users and policies change frequently in dynamic environments, storing the XACML policy as an XML file is not an optimal way for managing the policies. In future, as more and more organizations adopt the XACML standard it is important to address these issues and design a new system that can create, store and manage the XACML policies efficiently. As Grid involves several organizations, the distributed storage and administration of the XACML policies has to be supported.

Privacy protection is another important challenge in today's environment. Users may not want to reveal their personal information to the resource providers. Lots of research has been performed in this area, such as the use of anonymization techniques, which protect the identity of users in data publishing. But, they do not allow users to dynamically control the information being released to resource providers. Also, several

resource providers do not support anonymous user access. The use of RBAC and Shibboleth in my dissertation work addresses the privacy protection problem to certain extent. Shibboleth supports pseudonymous user access by mapping the identity to a pseudo-identity. However, this information is stored in a file which is not a secure way. In our system using RBAC, policies are specified based on the roles of the user instead of the individual identities. This protects the privacy of the user, as authorization is performed with user roles instead of user identities. However, the correlation between the user and roles can be easily identified by attackers by eavesdropping and phishing. This issue can be solved by introducing user session-based pseudo-identity with Shibboleth.

Geospatial ontology [37] and time ontology [26] can be used to specify the access control policies based on the location of the user, the resource requested, and the time at which the request is made. Geospatial ontology represents geospatial concepts and properties of the Web resources. Time ontology in OWL provides a vocabulary for expressing facts about topological relations among instants and intervals, together with information about durations, and about dates and times. For example, specification of access control policies using 'next-door-to' and GPS tracking data to allow the user access to a specific resource is not possible with the current technologies. Additionally, Geospatial ontology is still an emerging standard and requires further research. While most of the proposed work is targeted towards improving the access control system in Data Grids, the proposed systems can be extended to other environments, such as cyber-physical systems and cloud computing systems.

Bibliography

[1] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Gianoli, F. Spataro, et al., "Managing Dynamic User Communities in a Grid of Autonomous Resources," *Proc. of Int'l Conf. for Computing in High Energy and Nuclear Physics*, 2003.

[2] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. P. Chue Hong, P. Dantressangle, A. C. Hume, et al., "OGSA-DAI Status and Benchmarks," *Proc. of the UK e-Science All Hands Meeting*, 2005.

[3] M. Atkinson, K. Karasavvas, M. Antonioletti, R. Baxter, A. Borley, N. Chue Hong, A. Hume, et al., "A New Architecture for OGSA-DAI," *Proc. of the UK e-Science All Hands Meeting*, 2005.

[4] M. Baker, A. Apon, C. Ferner, and J. Brown, "Emerging Grid Standards," Computer, vol. 38, no. 4, pp. 43–50, 2005.

[5] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC Storage Resource Broker," *Proc. of CASCON*, 1998.

[6] C. Baru, A. Rajasekar, "A hierarchical access control scheme for digital libraries," *Proc. of the 3rd ACM Conference on Digital Libraries*, pp. 275–276, 1998.

[7] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American Magazine*, May, 2001.

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

[9] http://www.globus.org/toolkit/docs/4.0/security/key-index.html.

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

[11] A. Chakrabarti, "Grid Authorization Systems," in Grid Computing Security, Springer-Verlag, pp. 67–104, 2007.

[12] F. J. G. Clemente, G. M. Perez, J. A. B. Blaya, A. F. G. Skarmeta, "Description of Policies Enriched by Semantics for Security Management," in Web Semantics Ontology, D. Taniar and J. Rahayu (Eds.), Idea Group Publishing, pp. 365-391, 2006.

[13] E. Damiani, S. De Capitani di Vimercati, C. Fugazza, and P. Samarati, "Extending Policy Languages to the Semantic Web," *Proc. of the 4th International Conference on Web Engineering*, pp. 330–343, 2004.

[14] J. Davies, R. Studer, and P. Warren (Eds.), *Semantic Web Technologies: Trends and Research in Ontology-based Systems*, Wiley, 2006.

[15] Y. Demchenko, C. de Laat, L. Gommans, R. van Buuren, "Domain based access control model for distributed collaborative applications," *Proc. of the 2nd IEEE International Conference on e-Science and Grid Computing*, 2006.

[16] D. Ferraiolo and R. Kuhn, "Role-based Access Control," *Proc. of the 15th National Computer Security Conference*, 1992.

[17] 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. on Information and System Security*, vol. 2, no. 1, pp. 34–64, 1999.

[18] I. Foster and C. Kesselman, "The Globus Toolkit," in The Grid: Blueprint for a New Computing Infrastructure, I. Foster and C. Kesselman (Eds.), Morgan Kaufman, pp. 259–278, 1999.

[19] I. Foster and C. Kesselman, "Security, Accounting, and Assurance," in *The Grid: Blueprint for a New Computing Infrastructure,* I. Foster, C. Kesselman (Eds.), Morgan Kaufmann, pp. 395–420, 1999.

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

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

[22] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," Journal of Computer Science and Technology, vol. 22, no. 4, pp. 513–520, 2006.

[23] E. Freudenthal, T. Pesin, L. Port, E. Keenan, V. Karamcheti, "dRBAC: distributed role-based access control for dynamic coalition environments," *Proc. of the 22nd Int'l Conference on Distributed Computing Systems*, pp. 411–420, 2002.

[24] The Globus Security Team, "Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective," available at http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf, 2005.

[25] T. Gruber, "A translation approach to portable ontologies," *Knowledge Acquisition* 5(2):199–220, http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html.

[26] J. Hobbs and F. Pan (Eds.), "Time Ontology in OWL," available at http://www.w3.org/TR/owl-time/.

[27] I. Horrocks, P. Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," http://www.w3.org/Submission/SWRL/, 2004.

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

[29] M. Humphrey, G. Wasson, J. Gawor, et al., "State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations," *Proc. of the 14th IEEE International Symposium on High Performance Distributed Computing*, 2005.

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

[31] *Jena – A Semantic Web Framework for Java*, http://jena.sourceforge.net/index.html.

[32] H. Jin, W. Qiang, X. Shi, and D. Zou, "RB-GACA: A RBAC Based Grid Access Control Architecture," *Int'l Journal of Grid and Utility Computing*, vol. 1, no. 1, pp. 61–70, 2005.

[33] 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, 2004.

[34] J. B. D. Joshi, E. Bertino, U. Latif, A. Ghafoor, "A generalized temporal role-based access control model," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 4–23, 2005.

[35] L. Kagal, T. Finin, and A. Joshi, "A Policy Based Approach to Security for the Semantic Web," *Proc. of the 2nd Int'l Semantic Web Conf. (ISWC)*, 2003.

[36] H. K. Lee, H. Luedemann, "A lightweight decentralized authorization model for inter-domain collaborations," *Proc. of the ACM Workshop on Secure Web Services*, pp. 83–89, 2007.

[37] J. Lieberman, R. Singh, C. Goad, "W3C Geospatial Ontologies," available at http://www.w3.org/2005/Incubator/geo/XGR-geo-ont-20071023/.

[38] M. Lorch, S. Proctor, R. Lepro, D. Kafura, S. Shah, "First experiences using XACML for access control in distributed systems," *Proc. of the ACM Workshop on XML Security,* pp. 25–37, 2003.

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

[40] T. Mayfield, J. E. Roskos, S. R. Welke, and J. M. Boone, "Integrity in Automated Information Systems," Technical Report, National Computer Security Center, 1991.

[41] MCAT. Available via http://www.sdsc.edu/srb/index.php/MCAT

[42] P. Mitra, C. Pan, P. Liu, V. Atluri, "Privacy-preserving Semantic Interoperation and Access Control of Heterogenous Databases," *Proc. of the ACM Symposium on Information, Computer and Communications Security,* pp. 66–77, 2006.

[43] V. Muppavarapu and S. M. Chung, "Semantic-Based Access Control for Grid Data Resources in Open Grid Services Architecture - Data Access and Integration (OGSA-DAI)," *Proc. of the 20th IEEE Int'l Conf. on Tools with Artificial Intelligence (ICTAI 2008)*, vol. 2, pp. 315–322, 2008.

[44] V. Muppavarapu and S. M. Chung, "Role-Based Access Control in a Data Grid Using the Storage Resource Broker and Shibboleth," *Journal of Grid Computing*, vol. 7, no. 2, Springer, pp. 265–283, 2009.

[45] 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.

[46] Organization for the Advancement of Structured Information Standards (OASIS), *ebXML Registry Technical Committee*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=regrep.

[47] Organization for the Advancement of Structured Information Standards (OASIS), *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V1.1*, available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, 2003.

[48] Organization for the Advancement of Structured Information Standards (OASIS), *Core and hierarchical role based access control (RBAC) profile of XACML v2.0*, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf.

[49] Organization for the Advancement of Structured Information Standards (OASIS), *eXtensible Access Control Markup Language (XACML) Version 2.0*, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, 2005.

[50] Organization for the Advancement of Structured Information Standards (OASIS), *ebXML registry information model version 3.0*, http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rim-3.0-os.pdf (2005)

[51] Organization for the Advancement of Structured Information Standards (OASIS), *SAML 2.0 profile of XACML v2.0*, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf (2005)

[52] Organization for the Advancement of Structured Information Standards (OASIS), *eXtensible Access Control Markup Language (XACML) Version 3.0*, http://www.oasis-open.org/committees/download.php/28318/xacml-3.0-core-wd-06.zip, 2008.

[53] Object, Metadata and Artifacts Registry. Available via http://ebxmlrr.sourceforge.net/3.0/

[54] S. Otenko and D. Chadwick, "A Comparison of the Akenti and PERMIS Authorization Infrastructures," available at http://sec.isi.salford.ac.uk/download/AkentiPERMISDeskComparison2-1.pdf, 2003.

[55] *OWL Web Ontology Language Guide*, http://www.w3.org/TR/55/, 2004.

[56] *OWL Web Ontology Language Overview*, http://www.w3.org/TR/owl-features/, 2004.

[57] C. Pan, P. Mitra, and P. Liu, "Semantic access control for information interoperation," *Proc. of 11th ACM Symposium on Access Control Models and Technologies*, 2006, pp. 237–246.

[58] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke., "A Community Authorization Service for Group Collaboration," *Proc. of the 3rd IEEE Int'l Workshop on Policies for Distributed Systems and Networks*, 2002.

[59] *Pellet*, http://pellet.owldl.com/.

[60] A. L. Pereira, V. Muppavarapu, and S. M. Chung, "Role-Based Access Control for Grid Database Services Using the Community Authorization Service," *IEEE Trans. on Dependable and Secure Computing*, vol. 3, no. 2, 2006, pp. 156–166.

[61] A. L. Pereira, V. Muppavarapu, and S. M. Chung, "Managing Role-Based Access Control Policies for Grid Databases in OGSA-DAI Using CAS," *Journal of Grid Computing*, vol. 5, no. 1, 2007, pp. 65–81.

[62] T. Priebe, W. Dobmeier, and N. Kamprath, "Supporting Attribute-based Access Control with Ontologies," *Proc. of the 1st Int'l Conf. on Availability, Reliability and Security (ARES)*, pp. 465–472, 2006.

[63] A. Rajasekar, M. Wan, R. Moore, "MySRB & SRB: components of a data Grid," *Proc. of the 11th IEEE Int'l Symposium on High Performance Distributed Computing*, pp. 301–310, 2002.

[64] A. Rajasekar, M. Wan, R. Moore, et al., "Storage Resource Broker – Managing Distributed Data in a Grid," *Computer Society of India Journal*, vol. 33, no. 4, 2003.

[65] C. Ramaswamy and R. S. Sandhu, "Role-based Access Control Features in Commercial Database Management Systems," *Proc. of the 21st National Information Systems Security Conference*, 1998.

[66] *Resource Description Framework (RDF): Concepts and Abstract Syntax*, http://www.w3.org/TR/rdf-concepts/, 2004.

[67] *RDF Vocabulary Description Language 1.0: RDF Schema*, http://www.w3.org/TR/rdf-schema/, 2004.

[68] *Rei: A Policy Specification Language*, http://rei.umbc.edu/.

[69] *The Rule Markup Initiative*, http://www.ruleml.org/.

[70] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based Access Control Models," *IEEE Computer*, vol. 29, no. 2, 1996, pp. 38–47.

[71] R. Sandhu, V. Bhamidipati, Q. Munawer, "The ARBAC97 model for role-based administration of roles," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 105–135, 1999.

[72] R. Sandhu, D. F. Ferraiolo, and D. R. Kuhn, "The NIST Model for Role Based Access Control: Towards a Unified Standard," *Proc. of the 5th ACM Workshop on Role Based Access Control*, 2000.

[73] T. Scavo, V. Welch, "A Grid authorization model for science gateways," *International Workshop on Grid Computing Environments*, 2007.

[74] Secretariat of Information Technology Industry Council (ITI): American National Standard for Information Technology – Role based access control. Available via http://csrc.nist.gov/rbac/rbac-std-ncits.pdf (2003)

[75] L. Seitz, J. Pierson, and L. Brunie, "Semantic Access Control for Medical Applications in Grid Environments," *Proc. of European Conference on Parallel Processing*, pp. 374–383, 2003.

[76] *Sesame*, http://openrdf.org/doc/sesame2/2.1.3/users/userguide.html.

[77] "Shibboleth: Web Single Sign-On and Federating Software," http://www.internet2.edu/pubs/shibboleth-infosheet.pdf, 2008.

[78] "SICSACML: XACML 3.0," http://www.sics.se/node/2465

[79] *SPARQL Query Language for RDF*, http://www.w3.org/TR/rdf-sparql-query/, 2008.

[80] "Sun's XACML Implementation," http://sunxacml.sourceforge.net/guide.html, 2004.

[81] *Semantic Web Services Language*, http://www.w3.org/Submission/SWSF-SWSL/, 2005.

[82] 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. of Int'l Conf. for Computing in High Energy and Nuclear Physics*, 2003.

[83] A. Toninelli, J. M. Bradshaw, L. Kagal, and R. Montanari, "Rule-based and Ontology-based Policies: Toward a Hybrid Approach to Control Agents in Pervasive Environments," *Proc. of the Semantic Web and Policy Workshop*, 2005.

[84] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok, "Semantic Web Policy Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder," *Proc. of the 2nd Int'l Semantic Web Conf. (ISWC)*, 2003.

[85] A. Uszok et al., "KaoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement," *Proc. IEEE 4th Int'l Workshop Policies for Distributed Systems and Networks,* pp. 93–96, 2003.

[86] A. Uszok, J. M. Bradshaw, and R. Jeffers, "KAoS: A Policy and Domain Services Framework for Grid Computing and Semantic Web Services," *Proc. of iTrust 2004*, LNCS 2995, Springer-Verlag, pp. 16–26, 2004.

[87] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke, "Security for Grid services," *Proc. of the 12th International Symposium on High-Performance Distributed Computing*, Seattle, WA, pp. 48–57, 2003.

[88] V. Welch, T. Barton, K. Keahey, and F. Siebenlist, "Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration," *Proc. of the 4th Annual PKI R&D Workshop*, 2005.

[90] M. Yague, A. Mana, L. Lopez, and J. M. Troya, "Applying the Semantic Web Layers to Access Control," *Proc. of the Database and Expert Systems Applications Workshop on Web Semantics (WebS 2003)*, 2003.

[91] M. Yague, M. Gallardo, and A. Mana, "Semantic Access Control Model: A Formal Specification," *Proc. of European Symposium on Research in Computer Security*, LNCS 3679, Springer-Verlag, pp. 24–43, 2005.

[92] K. Yee, "Secure Interaction Design and the Principle of Least Authority," *Proc. of Workshop on Human-Computer Interaction and Security Systems*, 2003.

[93] G. Zhang and M. Parasher, "Dynamic Context-Aware Access Control for Grid Applications," *Proc. of the 4th Int'l Workshop on Grid Computing*, pp. 101–108, 2003.

[94] V. Muppavarapu, A. L. Pereira, and S. M. Chung, “Role-Based Access Control for a Grid System Using OGSA-DAI and Shibboleth,” to appear in *The Journal of Super Computing*, 2009.

[95] M. Cannataro and D. Talia, “The Knowledge Grid,” Communications of the ACM, vol. 46, no. 1, pp. 89–93, 2003.